Spiking Neural P Systems and Kernel P Systems

Marian Gheorghe[1], Florentin Ipate[2], Kumar Kannan[3], Savas Konur[1], Lakshmanan Kuppusamy[3*], Raluca Lefticaru[1], Anand Mahendran[4], Mihai Ionuţ Niculescu[2]

[1]School of Computer Science and Electronics, University of Bradford, West Yorkshire, BD7 1DP, United Kingdom.
[2]Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest, Str Academiei 14, Romania.
[3]School of Computer Science and Engineering, VIT, Vellore, 632014, Tamilnadu, India.
[4]School of Computer Science and Engineering, VIT, Chennai, 600127, Tamilnadu, India.

*Corresponding author(s). E-mail(s): klakshma@vit.ac.in;
Contributing authors: m.gheorghe@bradford.ac.uk; ipate@gmail.com; kkumar@vit.ac.in; s.konur@bradford.ac.uk; r.lefticaru@bradford.ac.uk; manand@vit.ac.in; ionutmihainiculescu@gmail.com;

Abstract

Spiking neural P systems represent one the most dynamic and active research area of membrane computing, with many variants being defined and investigated, and a broad spectrum of applications reported. Often, each newly introduced spiking neural P system is compared with others to reveal its potential. However, there is a need to compare them from another perspective, when they are all represented with the same instrumentation within a given framework and where quantitative metrics can precisely describe the complexity of their representations. In this paper a selection of the best known and investigated classes of spiking neural P systems are mapped into kernel P system models and different encodings of the computation results and strategies of using the rules are investigated. Complexity metrics associated with the translation processes are assessed. Specific methods to check model correctness and to provide test sets for implementations are used for an example in order to illustrate the connection between these P systems.

Keywords: membrane computing, spiking neural P systems, kernel P systems, descriptional and time complexity, testing, verification, model checking, kPWorkbench.

1 Introduction

Membrane computing, a research field initiated by Gh. Păun [34, 35], is a computing paradigm inspired by the structure and bio-chemical interactions of the living cells. The key models are called *membrane systems* or *P systems*. The field has developed very fast on various research directions and many classes of membrane systems (P systems) have been investigated. These may be classified as *cell-like*, *tissue-like* and *neural-like* P systems. A presentation of the most important membrane computing models is available in [36, 37] and more recently in a survey paper [48].

A class of neural-like P systems, called *Spiking Neural P systems* (*SN P systems*, for short), has been introduced in [14], inspired by the neurophysiological behaviour of neurons (in brain) sending electrical impulses along axons to other neurons. Significant results on SN P systems have been reported in the literature. A multitude of SN P system models have been considered and investigated with features such as: anti-spikes and inhibitory synapses [28], inhibitory rules [40], astrocytes [30], polarizations [38, 58], request rules [50, 56], multiple channels [46], extended channel rules [25], colored spikes [52], white hole neurons [49], lateral inhibition [24], rules on synapses [51], weights and potential (threshold) [55], dynamic threshold [44], weights on synapses and extended rules with delay [32], weights and delays on synapses [23], dendrites [39], plasticity [3, 26], with neuron division and budding[29], scheduled synapses [2]; or new classes of systems: axon P systems in [62], fuzzy reasoning SN P systems [45], coupled SNP [42], numerical SN P systems [57], nonlinear SN P systems [41]. Various research topics based on SN P system models have been considered. A bibliography of SN P systems is available in [31], a survey of the main classes of SN P systems is presented in [22] and of learning SN P systems is described in [5]. Theoretical results and applications of SN P systems are discussed in [47]. In 2024 two research textbooks on SN P systems have been published [43, 61]. For several of the above mentioned SN P systems have been considered different strategies for using the rules as well as ways of encoding the results of the computation - the key topics in this respect have been presented in [22].

Another type of P systems, called *kernel P systems* (*kP systems*, for short), has been introduced in [9] in order to capture in a unified manner features of existing membrane computing models together with new concepts, such as types, guarded rules using Boolean conditions, user defined execution strategy for each component type. All these make the kP system model more amenable for describing different problems occurring in various areas, from specific computer science topics, such as communication and synchronisation [8], to applications in synthetic biology [19, 18]. kP systems have been conceived as a membrane computing model allowing to specify problems, verify the models correctness and test the implementations. They are supported by an expressive domain specific formal language allowing the models to be simulated with a software framework, called kPWorkbench [1], which also includes a verification component [11]. A detailed presentation of this tool has been made in [20]. The modelling, simulation, verification and testing aspects of kP systems have been presented in [7, 8].

In this paper we investigate, for several classes of SN P systems using regular expressions that can be expressed with features of kP systems, how these are mapped into kP systems exhibiting the same behaviour. The classes of SN P systems considered are standard and extended SN P systems with and with no delay, with colored spikes, multiple channels, anti-spikes, polarizations, weights on synapses, potential and threshold. Two types of encoding the computation results and various strategies of using the rules that are defined and investigated for SN P systems are studied in the context of kP systems as well. A logical gate example illustrates how different SN P systems and their kP system counterparts can model it. Finally, a short presentation on how testing and verification methods developed for kP systems can be used in the context of using SN P systems to model the logical gate example.

Previously, a formal framework for specifying different types of SN P systems has been devised [54]. This approach shows how the same formal environment capture specific features of different types of SN P systems. Our investigation allows to define for each of the SN P systems investigated algorithms for translating such systems into corresponding kP systems. In our approach, metrics to assess the impact of various features of different classes of SN P systems on the complexity of their kP system representations are presented. Moreover, the instrumentation provided by the kP system environment facilitates the verification of the model correctness, through model checking methods, of different SN P systems and the automated testing of their implementations.

This paper is structured as follows: Section 2 presents some preliminary concepts and basic definitions of SN P and kP systems. Section 3 provides the key results and outcomes of the paper, presenting the mapping of different classes of SN P systems into corresponding kP systems and how different encodings of the results and strategies of using the rules of SN P systems are represented in the context of kP systems; complexity metrics associated with the mapping processes involved are investigated. For the logical gate example modelled with an SN P system, a testing approach for validating this application is presented. This uses a test set derived from the inferred X-machine obtained from the computation sequences, up to a certain limit, of the kP system model (that corresponds to the SN P system modelling the logical gate) is presented in Section 4, together with the formal verification, through model checking, of some properties of the SN P system. Finally, conclusions are presented in Section 5.

2 Preliminaries and basic definitions

Some concepts, notations and basic definitions used subsequently in this paper are introduced in this section. More formal language theory and membrane computing concepts and results can be found in [36, 37].

For a finite alphabet $A = \{a_1, \cdots, a_p\}$, A^* denotes the set of all strings (sequences) over A. The empty string is denoted by λ and $A^+ = A^* \setminus \{\lambda\}$ denotes the set of non-empty strings.

A regular expression over an alphabet A is defined as follows: (i) λ and each $a \in A$ is a regular expression; (ii) if E_1, E_2 are regular expressions over A, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$ and $(E_1)^+$ are regular expressions over A; and (iii) nothing else is a regular expression over A. Parentheses, "(" and ")", are not in A, they are used to indicate the order of applying the operators of the regular expression. When no confusion may appear, they can be omitted. $E_1^+ \cup \lambda$ can be written as E_1^*. With each regular expression E, a regular language $L(E)$ is associated as follows: (i) $L(\lambda) = \lambda$, and $L(a) = \{a\}$, for $a \in A$; (ii) $L((E_1)(E_2)) = L(E_1)L(E_2)$, $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$ and $L((E_1)^+) = L(E_1)^+$, for any regular expressions E_1, E_2.

A multiset over A is a mapping $f : A \to \mathbb{N}$, represented as a string $a_1^{f(a_1)} \cdots a_p^{f(a_p)}$, where the order is not important, and the elements which are not in the support of f (i.e., elements a_j, $1 \le j \le p$, having $f(a_j) = 0$) are omitted. In the sequel, multisets will be represented by such strings.

The standard and extended *Spiking Neural P systems* (*SN P systems*) are defined below. Later, other variants of SN P systems will be introduced by only presenting their key features distinguishing them from the previous ones. A restricted class of *kernel P systems* (*kP systems*) is also presented in this section as well..

2.1 Standard and extended SN P systems

The definition of a standard SN P system is given in [14].

Definition 1 A *Spiking Neural P system* of degree m, $m \ge 1$, is a tuple

$$\Pi = (O, \sigma_1, \cdots, \sigma_m, syn, i_0),$$

where

- $O = \{a\}$ is a singleton alphabet (a is called *spike*);
- σ_i is a neuron, $\sigma_i = (n_i, R_i), 1 \le i \le m$, where
  - $n_i \ge 0$ is the *initial number* of spikes from σ_i;
  - R_i is a finite set of *rules* of the following two forms:
    - $E/a^c \to a; d$ where E is a regular expression over $\{a\}$, and $c \ge 1$, $d \ge 0$ are natural numbers (*spiking* or *firing* rules);
    - $a^s \to \lambda$, where $s \ge 1$ is a natural number, such that for any spiking rule $E/a^c \to a$ from R_i, $a^s \notin L(E)$ (*forgetting* rules);
- $syn = \{(i, j) | 1 \le i, j \le m, i \ne j\}$ (*synapses* between distinct neurons);
- $i_0 \in \{1, \ldots m\}$ indicates the *output* neuron.

The system evolves in steps marked by a global discrete clock. At each time step, in each neuron having enabled rules, only one of them must fire/be applied. If more than a rule is enabled then one of them is non-deterministically chosen and is applied.

A rule $E/a^c \to a; d$ is enabled (can fire), in neuron σ_i at time step t, if σ_i contains n spikes, such that $a^n \in L(E)$ and $c \leq n$. When the rule fires, c spikes are removed from σ_i and one spike is sent to each neuron σ_j, linked through a synapse to σ_i, i.e., $(i, j) \in syn$, with delay d. Spikes sent out from the output neuron arrives into the environment. If $d = 0$ then the spike is released to all neurons σ_j at moment t, otherwise the spike will be released to all σ_j at time step $t + d$. In between, i.e., at times $t, \cdots, t + d - 1$, the neuron σ_i is *closed* and during that time it won't send and receive spikes. Every spike sent to a closed neuron is lost. At time step $t + d$, σ_i becomes *open* again, receiving spikes and sending those not released at time step t. A forgetting rule, $a^s \to \lambda$, can be applied only when the neuron contains exactly s spikes.

A *configuration* of the SN P system Π at step time h is denoted by $S_h = (S_{h,1}, \cdots, S_{h,m})$, where $S_{h,j} = a_j/t_j$, a_j representing the number of spikes in neuron σ_j and t_j, $1 \leq j \leq m$, indicating the number of time steps the neuron is closed.

Given two configurations S_h and $S_{h'}$ of the SN P system Π, a *transition*, denoted $S_h \Longrightarrow S_{h'}$, defines the process of obtaining $S_{h'}$ from S_h, where each component $S_{h',j}$, $1 \leq j \leq m$, of $S_{h'}$ is obtained from $S_{h,j}$ by applying at most one rule from σ_j, from those which are applicable, if any, to $S_{h,j}$.

A *computation* is a sequence of transitions, $S_0 \Longrightarrow \cdots \Longrightarrow S_t \Longrightarrow S_{t+1} \cdots$ The initial configuration is $S_0 = (n_1/0, \cdots, n_m/0)$. A *halting* computation ends in an open configuration where no rule is applicable.

The definition of an *extended SN P system* is given below [4].

Definition 2 An *extended Spiking Neural P system* of degree m, $m \geq 1$, is a tuple

$$\Pi = (O, \sigma_1, \cdots, \sigma_m, syn, i_0),$$

where, as in Definition 1, O is an alphabet, syn the set of synapses, i_0 the output neuron and $\sigma_i = (n_i, R_i)$, $i \leq i \leq m$, a neuron with n_i, $n_i \geq 0$, spikes and a finite set of rules, R_i. The rules have the form $E/a^c \to a^p$, where E is a regular expression over $\{a\}$, $c \geq 1$, and $p \geq 0$, are natural numbers and $c \geq p$.

A rule of a neuron σ_i of an extended SN P system is enabled in the same way with a rule of an SN P system, but it sends p, $p \geq 0$, spikes to each of the neurons σ_j linked through synapses $(i, j) \in syn$. If the rules in the above definition include a delay, d, then they are generalizations of spiking and forgetting rules of the standard SN P systems. When $p = 0$ a forgetting rule is obtained, where c objects are removed from the neuron.

The *result* of a standard (or an extended) SN P system can be defined in various ways. Below we refer to the generative case, selecting two situations as presented in [14, 22]. The result is obtained by counting either (i) the number of spikes received by the output neuron at the end of a halting computation, or sent to the environment during a halting computation, or (ii) the number of time steps executed by the SN P system between the first two spikes sent out to the environment by the output neuron. The type (i) of encoding the result of a halting computation will be considered for all the SN P systems investigated in the paper, whereas type (ii) will be applied only for standard and extended SN P systems.

For an SN P system, Π, one denotes by $\mathcal{N}(\Pi)$ all the results produced by Π.

In some circumstances SN P systems with one or more *input neurons* will be considered. In these cases these input neurons will be fed with spikes from the environment.

2.2 Kernel P systems

A formal definition of a simple version of kernel P system (kP system) is given below – more details regarding this model as well as its complete definition are available from [9].

Any P system consists of a set of membranes (regions) connected in a certain way. In a kP system each membrane, called *compartment*, is obtained (instantiated) from a more general entity, called *compartment type*. Each compartment type, t, is given by a *finite set of rules*, R, and an *execution strategy*, δ, i.e., $t = (R, \delta)$.

The majority of the P systems, including SN P systems, introduced above, execute the rules using a strategy that is applied in all membranes – a survey of such strategies of using the rules is presented in [22]. In contrast to this, kP systems allow each compartment to use a specific way of executing the rules.

T is a *set of compartment types*, $T = \{t_1, \cdots, t_s\}$, where $t_i = (R_i, \delta_i)$, $1 \leq i \leq s$, consists of a set of rules, R_i, and an execution strategy, δ_i, defined over R_i.

Given a set of compartment types, $T = \{t_1, \cdots, t_s\}$, a kP system is defined below.

Definition 3 A *kernel P system* of degree n, $n \geq 1$, is a tuple

$$k\Pi = (A, \mu, C_1, \cdots, C_n, i_0),$$

where

- A is a finite set of elements called *objects*;
- μ defines the *membrane structure*, which is an unoriented graph, (V, L), where V is a set of vertices representing *compartments*, $V = \{C_1, \cdots, C_n\}$, and L is a set of edges connecting compartments, called *links* $L = \{\{C, C'\}|$ for some $C, C' \in V\}$;
- $C_i = (t_i, w_{i,0})$, $1 \leq i \leq n$, is a *compartment* consisting of a *compartment type*, t_i, $t_i \in T$, and an *initial multiset*, $w_{i,0}$, over A; the compartment type $t_i = (R_i, \delta_i)$, consists of a finite set of *rules*, R_i, and an *execution strategy*, δ_i;
- i_0 indicates the *output* compartment where the result is obtained.

The kP system introduced above has one single type of rules, called *rewriting and communication rule*. Such a rule has the form $x \to y(y_1, t_1) \cdots (y_h, t_h)$ $\{g\}$, where $h \geq 0$, $x \in A^+$, $y \in A^*$, $y_i \in A^+$, $1 \leq i \leq h$, and g represents a *guard*. Since now on, this will be called *rule*. Before presenting the way the rule operates, we describe its guard.

A guard is defined with relational operators $(<, \leq, =, \neq, \geq, >)$ and Boolean operators ($\neg$ (negation), $\wedge$ (conjunction) and $\vee$ (disjunction)). A guard is assessed as a Boolean condition for a given multiset, w, whenever the rule is checked on whether it is applicable or not to w. For the guards written only with relational operators, $= a^n$, $< b^m$, $\geq c^p$ and a multiset w, the Boolean conditions assessed are $|w|_a = n$, $|w|_b < m$, $|w|_c \geq p$, respectively, where $|w|_x$ means the number of objects x occurring in w. Hence, the Boolean conditions above are true when the multiset w has exactly n occurrences of a, less than m objects b, at least p objects c, respectively. More complex guards can be written using Boolean operators together with relational ones.

Given a compartment C obtained from a type $t = (R, \delta)$ and w the current multiset of C, a rule $r : x \to y(y_1, t_1) \cdots (y_h, t_h)\ \{g\}$ is applicable to w if w contains the multiset x and the guard (evaluated as a Boolean condition for w) is true. If the compartment C is connected with the compartments C_j of type t_j and $\{C, C_j\} \in L$, then x is removed from w, y is added to it and y_j is sent to C_j, $1 \leq j \leq h$. If more than one compartment of type t_j exists and is linked with C, then one of them is non-deterministically chosen to receive y_j.

For a compartment type $t = (R, \delta)$ from T and $r_1, \cdots, r_s \in R$, δ is one of the following execution strategies

- *choice (alternative)*, denoted $\{r_1, \cdots, r_s\}$, with the meaning: one of the rules applicable will be non-deterministically chosen and executed; if none is applicable then nothing is executed;
- *maximal parallelism*, denoted $\{r_1, \cdots, r_s\}^T$, means: the rules will be applied a maximal number of times; when the rules are applied an arbitrary number of times, denoted $\{r_1, \cdots, r_s\}^*$, the execution strategy is called *arbitrary execution*; these execution strategies have the meaning used for tissue P systems;
- *sequential,* denoted $B_1 \cdots B_k$, $k \geq 1$, where B_j, $1 \leq j \leq k$, is either a simple rule or any of the above execution strategies, means: executing $B_1, \cdots, B_k$ one by one in sequence, starting with B_1 and stopping after B_k or immediately after any of the B_j which is not executed.

A *configuration* of the kP system $k\Pi$ at time step h is a tuple $c_h = (c_{h,1}, \ldots, c_{h,n})$, where $c_{h,i} \in A^*$ is the multiset from compartment C_i, $1 \leq i \leq n$. The *initial configuration* is $c_0 = (w_{1,0}, \ldots, w_{n,0})$.

Given two configurations $c_h = (c_{h,1}, \ldots, c_{h,n})$ and $c_{h'} = (c_{h',1}, \ldots, c_{h',n})$ of the kP system $k\Pi$ and a multiset of rules M_i applicable to $c_{h,i}$ from C_i, using δ_i, $1 \leq i \leq n$, as execution strategy, a *transition*, denoted by $c_h \Longrightarrow^{(M_1,\ldots,M_n)} c_{h'}$, is the process of obtaining $c_{h'}$ from c_h by applying M_i to $c_{h,i}$, $1 \leq i \leq n$.

A *computation* in a kP system is a sequence of transitions $c_0 \Longrightarrow^{(M_1^0,\ldots,M_n^0)}$ $, \cdots, c_t \Longrightarrow^{(M_1^t,\ldots,M_n^t)} c_{t+1} \cdots$ The multisets of rules may be dropped when these are not necessary. A configuration is called *final configuration*, if no rule can be applied to it. A computation ending in a *final configuration* is called *halting* computation. The *result* of a halting computation is obtained by counting the *number of objects* appearing in the output compartment of a halting computation. *The kP system model do not use the environment.* Similar to the case of SN P systems, for a kP system, $k\Pi$, one denotes by $\mathcal{N}(k\Pi)$ all the results produced by $k\Pi$.

The kP system model may use *input compartments*, similar to input neurons. In such circumstances the input compartments use only their initial multisets, as there is no environment to provide inputs.

3 SN P systems and kP systems

In this section the following topics are presented: (a) the mapping of several SN P systems into kP systems, when the former use type (i) of encoding the result, i.e., by counting the number of spikes received by the output neuron at the end of a halting computation, or sent to the environment during a halting computation (in Subsection 3.1); and (b) the mapping of standard and extended SN P systems using rules with no delay and type (ii) of encoding their results, i.e., by counting the number of time steps executed by the SN P system between the first two spikes sent out to the environment by the output neuron, into kP systems (in Subsection 3.2). Complexity aspects related to these mappings will be presented at the end of each of these subsections.

We start by presenting informally some principles of building a kP system derived from an extended SN P system. As we do not present the way they compute and get the final results, we consider only SN P systems that do not use an environment. In the example that will follow immediately afterwards, it will be shown how the two systems compute the same results. The precise construction of the kP systems corresponding to various SN P systems, where environment may be considered, will be provided when formal proofs will be provided.

First, we identify the key structural elements of a basic or extended SN P system, that are common to all those considered in this paper, and show their corresponding counterparts in a kP system.

These key elements of an extended SN P system, as presented in Definition 2, but having no delay attached to rules, are the neurons, their connections, called synapses, and the rules. These are presented in Table 1 together with their counterparts in a kP system.

	SN P system	kP system
Neurons - Compartments	$\sigma_i = (n_i, R_i)$ $1 \leq i \leq m$	$t_i = (R'_i, \delta_i), 1 \leq i \leq m$ $C_i = (t_i, w_{i,0}), w_{i,0} = a^{n_i}$
Synapses - Links	$(i, j) \in syn$ $1 \leq i, j \leq m; i \neq j$	$\{C_i, C_j\} \in L$ $1 \leq i, j \leq m; i \neq j$
Rules	$r \in R_i, 1 \leq i \leq m$ $r : E/a^c \rightarrow a^p; c \geq p > 0, c \geq 1$	$r' \in R'_i, 1 \leq i \leq m$ $r' : a^c \rightarrow (a^p, t_{i_1}) \ldots (a^p, t_{i_{h_i}})\{g_E\}$

Table 1: SN P system vs kP system structural elements

We observe on the first line of the table that for each neuron σ_i, $1 \leq i \leq m$, given by the initial number of spikes, n_i, represented in the system by a^{n_i}, and a set of rules, R_i, the corresponding elements in the kP system are a compartment, C_i, obtained from a type t_i, and having as initial multiset $w_{i,0} = a^{n_i}$. Each type is given by a set of rules R'_i and an execution strategy, δ_i, which is *choice*, corresponding to the way the SN P system executes the rules in each neuron, where only one of the executable rules is selected to run. As there is no environment for kP system models, when an *SN P system uses the environment, then the corresponding kP system will have one more compartment* – see the proof of Theorem 1.

On the second line it is shown that for each synapse between two distinct neurons, there is a link between the corresponding compartments of the kP system.

The third line of the table describes how the corresponding rules of the kP system are constructed in each compartment. Each rule r of R_i, $1 \leq i \leq m$, may be executed when the current multiset of the neuron satisfies the regular expression E which is matched by the guard g_E of r' from R'_i. When the regular expression is absent the rule r may be executed if the current multiset of the neuron is exactly a^c; in this case the guard of r' becomes $= a^c$. For the rules in Definition 2, one can have $p = 0$ as well. In this case the rule r' becomes $a^c \rightarrow \lambda \ \{g_E\}$ or $a^c \rightarrow \lambda \ \{= a^c\}$, depending on whether r has a regular expression, E, or not. As through the execution of r, the spikes a^p, $p \geq 1$, are sent to all neurons, σ_{i_e}, given that $(i, i_e) \in syn$, $1 \leq e \leq h_i$, then in the kP system the rule r' will send the objects a^p to all the compartments corresponding to the neurons σ_{i_e}. When $p = 0$, r is a forgetting rule and r' does not send anything to other compartments, it only removes a^c from C_i.

The example given below, a logical gate, presents a standard SN P system with no delay, as well as the equivalent kP system, illustrating the principles of building the kP system mentioned above.

The P systems presented in the next three examples are executed with the following tools: the SN P systems using P-Lingua (for a presentation see [60], Chapter 2) and the kP systems using kP-Lingua from kPWorkbench [1]. The computations produced by these tools are summarized as tables, but also available, as traces of execution, from the repository [27].

Example 1 Let $\Pi_1 = (\{a\}, \sigma_1, \cdots, \sigma_{10}, syn, 10)$, be a standard SN P system using rules with no delay, i.e., $d = 0$, as shown in Fig. 1. The sets of rules are $R_i = \{r_{1,i} : a \rightarrow a\}$, $i = 1, 2, 3, 6, 8$; $R_i = \{r_{1,i} : a \rightarrow a, r_{2,i} : a^2 \rightarrow a\}$, $i = 4, 7$; $R_i = \{r_{1,i} : a^2 \rightarrow a, r_{2,i} : a \rightarrow \lambda\}$, $i = 5, 9$, and $R_{10} = \emptyset$. The result of the computation, the number of spikes, is provided in the output neuron, σ_{10}. This SN P system describes a logical gate computing the result of the logical expression $((x \vee y) \wedge z) \wedge (x \vee z)$. The values 1 or 0 for the variables x, y, z, will be provided in the input neurons $\sigma_1, \sigma_2, \sigma_3$, respectively, either as initial number of spikes or entering from the environment. A spike present or entering an input neuron is considered 1 and no spike means 0. The following logical sub-expressions are computed: $x \vee y$ in σ_4, $x \vee z$ in σ_7, $(x \vee y) \wedge z$ in σ_5, and the given logical expression in σ_9. The final result, either 1 or 0, is obtained in σ_{10}. The neurons σ_6 and σ_8 are meant to synchronize the outputs from various sub-expressions utilised in the next computation step.

The kP system built from the SN P system Π_1, according to the principles mentioned above and illustrated by Table 1, is presented below.

The kP system $k\Pi_1 = (\{a\}, \mu, C_1, \cdots C_{10}, 10)$, is given by

1. $\mu = (V, L)$, and $V = \{C_1, \cdots C_{10}\}$ and $L = \{\{C_1, C_4\}, \{C_1, C_7\}, \{C_2, C_4\}, \{C_3, C_6\}, \{C_3, C_7\}, \{C_4, C_5\}, \{C_5, C_9\}, \{C_6, C_5\}, \{C_7, C_8\}, \{C_8, C_9\}, \{C_9, C_{10}\}\}$;
2. each $C_i = (t_i, w_{i,0})$ is obtained from a compartment type, $t_i = (R'_i, \delta_i)$, $1 \leq i \leq 10$, with initial multisets $w_{1,0} = x, w_{2,0} = y, w_{3,0} = z$, where each of the x, y, z is either a or λ (equivalently, 1 or 0, respectively) and $w_{i,0} = \lambda$, $4 \leq i \leq 10$. Compartments C_1, C_2 and C_3 are input compartments with initial multisets mentioned above. The sets of rules are $R'_1 = \{r'_{1,1} : a \rightarrow (a, t_4)(a, t_7)\}$, $R'_2 = \{r'_{1,2} : a \rightarrow (a, t_4)\}$, $R'_3 = \{r'_{1,3} : a \rightarrow (a, t_6)(a, t_7)\}$, $R'_4 = \{r'_{1,4} : a \rightarrow (a, t_5) \ \{= a\}, r'_{2,4} : a^2 \rightarrow (a, t_5)\}$,

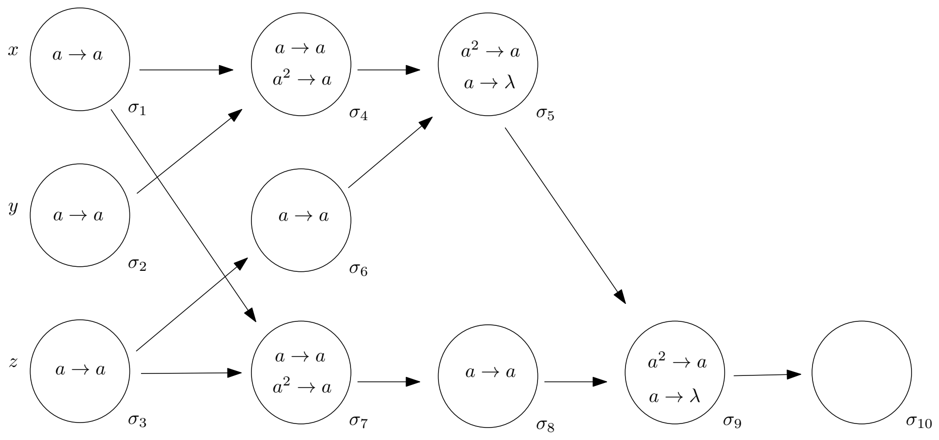

Fig. 1: SNP system with no delay, Π_1, for $((x \vee y) \wedge z) \wedge (x \vee z)$

$R'_5 = \{r'_{1,5} : a^2 \to (a, t_9), r'_{2,5} : a \to \lambda \ \{= a\}\}$, $R'_6 = \{r'_{1,6} : a \to (a, t_5)\}$, $R'_7 = \{r'_{1,7} : a \to (a, t_8) \ \{= a\}, r'_{2,7} : a^2 \to (a, t_8)\}$, $R'_8 = \{r'_{1,8} : a \to (a, t_9)\}$, $R'_9 = \{r'_{1,9} : a^2 \to (a, t_{10}), r'_{2,9} : a \to \lambda \ \{= a\}\}$, $R'_{10} = \emptyset$; each execution strategy, $\delta_i, 1 \le i \le 9$, is *choice* (*alternative*) and $\delta_{10} = \emptyset$.

Table 2 shows how the SN P system using only rules with no delay presented in Fig. 1 evolves from the start until the result is obtained for the input values $x = \lambda$ or a, $y = z = a$. When the content of a neuron has two values (for instance, a/a^2 in σ_4), the first value is obtained for the inputs $x = \lambda$, $y = z = a$ and the second for inputs $x = y = z = a$.

Steps	$\sigma_1(x)$	$\sigma_2(y)$	$\sigma_3(z)$	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}
0	λ/a	a	a	-	-	-	-	-	-	-
1	-	-	-	a/a^2	-	a	a/a^2	-	-	-
2	-	-	-	-	a^2	-	-	a	-	-
3	-	-	-	-	-	-	-	-	a^2	-
4	-	-	-	-	-	-	-	-	-	a

Table 2: SN P system with no delay, Π_1, sequence of steps

The table corresponding to $k\Pi_1$ is the same as Table 2, with C_i replacing σ_i, $1 \le i \le 10$.

3.1 Relationships between SN P systems and kP systems

Now, one can prove that any standard SN P system with no delay is mapped into a kP system having the same outputs; and similarly for an extended SN P system. Before proceeding with this investigation, a remark is made regarding the relationship between the regular expressions which appear in the rules of an SN P system and guards occurring in the rules of the corresponding kP system.

In order to map the SN P system rules into kP system rules, the regular expression that may be attached with an SN P system rule has to be transformed into a guard of the kP system rule. Some examples of regular expressions and the corresponding guards are described below.

First, let us consider some finite regular expressions: a, a^2 and $a^2 \cup a^5$. The corresponding guards are: $= a$, $= a^2$, and $= a^2 \vee = a^5$, respectively. For the infinite regular expressions, a^+, a^3a^+ and $a^2 \cup a^5a^*$, the corresponding guards are $\geq a, > a^3$, and $= a^2 \vee \geq a^5$, respectively. However, there are infinite regular expressions, such as $(a^k)^+$, which don't have corresponding guards, as some arithmetic expressions to denote multiples of k cannot be expressed with the current guard definition. There are ways to simulate infinite regular expressions such as $(a^k)^+$ within the context of kP systems with sequences of rules that extract blocks of k symbols from a multiset, but they increase significantly the complexity of these P systems and the solutions provided may require adaptations when different mappings are applied. Hence, they are not considered in the paper. However, with the current guards one can translate the simplified form of infinite regular expressions (i.e., a^+) used in the proof of Turing completeness [13]. There are also variants of SN P systems that do not use regular expressions, but some other conditions for selecting the rules, such as SN P systems with weights, potential and threshold [55], or with polarizations [38, 58]. These variants will be presented later in this section.

Remark 1 In the sequel we consider that each of the SN P systems considered uses in its rules only regular expressions that can be translated into a guard of the corresponding rule of the kP system.

Now, we focus on **standard/extended SN P systems using only rules with no delay**. A sketchy mapping of any standard SN P system, having rules with no delay, into a kP system is provided in [12]. In the proof of the next theorem the complete process of constructing a kP system for a given standard/extended SN P system using rules with no delay is described.

Theorem 1 *For any standard or extended SN P system, using only rules with no delay,* Π_e*, there exists a kP system,* $k\Pi_e$*, such that* $\mathcal{N}(\Pi_e) = \mathcal{N}(k\Pi_e)$*.*

Proof Given that a standard SN P system is a particular case of an extended one, the later will be considered below.

Let $\Pi_e = (O, \sigma_1, \ldots, \sigma_m, syn, i_0)$ be an extended SN P system, where $O = \{a\}$; $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, with $n_i \geq 0$, being the initial number of spikes in the neuron and R_i the set of rules of the form $E/a^c \to a^p$, where $c \geq 1$ and $c \geq p \geq 0$; syn the set of synapses between neurons; and i_o the output neuron. The following kP system, $k\Pi_e = (A, \mu, C_1, \cdots, C_n, i_0)$, described below, is constructed based on the principles illustrated by Table 1.

1. $A = O$.
2. For each neuron σ_i, $1 \leq i \leq m$, of Π_e, there is a compartment $C_i = (t_i, w_{i,0})$ in $k\Pi_e$ obtained from a compartment type $t_i = (R'_i, \delta_i)$, as described in Table 1.

   If Π_e uses the environment for sending spikes from the output neuron, then a compartment $C_{m+1} = (t_{m+1}, \lambda)$, where $t_{m+1} = (\emptyset, \emptyset)$, is added to $k\Pi_e$. This means that C_{m+1} has no rules and no execution strategy is needed. So, $n = m + 1$ or $n = m$, depending on whether the Π_e is using or not the environment.

3. $\mu = (V, L)$, where V is the set of compartments and L contains all the edges (links). The set V consists of all the compartments, C_i, $1 \le i \le n$, and L includes a link $\{C_i, C_j\} \in L$, if (i, j) or (j, i) is in *syn*. When the environment is used by Π_e, then a new link $\{C_{i_0}, C_{m+1}\}$ is added to L.
   (a) Let $r_{j,i} : E_{j,i}/a^{c_{j,i}} \to a^{p_{j,i}} \in R_i$, $1 \le j \le k_i$, be a rule of the neuron σ_i, $1 \le i \le m$, $i \ne i_0$, connected with σ_{i_e}, $1 \le e \le h_i$, $h_i \ge 1$ (i.e., $(i, i_e) \in syn$). If $p_{j,i} \ge 1$, then a rule $r'_{j,i} : a^{c_{j,i}} \to (a^{p_{j,i}}, t_{i_1}) \cdots (a^{p_{j,i}}, t_{i_{h_i}})$ $\{g_{E_{j,i}}\}$ is added to R'_i, the set of rules of type t_i, where $g_{E_{j,i}}$ is the guard obtained from the regular expression $E_{j,i}$. If $p_{j,i} = 0$, i.e., $r_{j,i}$ is a forgetting rule, then a rule $a^{c_{j,i}} \to \lambda$ $\{g_{E_{j,i}}\}$ is added to R'_i. In the case of the output neuron, σ_{i_0}, similar to the other neurons, for each rule $r_{j,i_0} : E_{j,i_0}/a^{c_{j,i_0}} \to a^{p_{j,i_0}} \in R_{i_0}$, $1 \le j \le k_{i_0}$, $p_{j,i_0} > 0$, in the compartment type t_{i_0} a rule r'_{j,i_0} is added to R'_{i_0} if Π_e does not use the environment. Otherwise, to the right hand side of $r'_{i_0,j}$ above it is added $(a^{p_{j,i_0}}, t_{m+1})$, which means that $a^{p_{j,i_0}}$ is sent to C_{m+1}. When $p_{j,i_0} = 0$ then the right hand side of r'_{j,i_0} is λ.
   (b) The execution strategy δ_i is *choice*.
   (c) $w_{i,0} = a^{n_i}$, $1 \le i \le m$.
4. The output compartment of the kP system is C_{i_0}.

One can show by induction that $S_0 \Longrightarrow S_1 \Longrightarrow \cdots \Longrightarrow S_t$ is a halting computation in Π_e, iff $c_0 \Longrightarrow c_1 \Longrightarrow \cdots \Longrightarrow c_t$ is a halting computation in $k\Pi_e$, with $S_h = (S_{h,1}, \cdots, S_{h,m}, S_{h,E})$ ($S_{h,E}$ appears only when the environment is used to get the result) and $c_h = (w_{h,1}, \cdots, w_{h,n})$ ($n = m$, when the environment is not used by Π_e and $n = m + 1$, otherwise), such that $a^{S_{h,i}} = w_{h,i}$, $1 \le i \le m$, $a^{S_{h,E}} = w_{h,m+1}$, $0 \le h \le t$.

When the environment is not used, then $S_{h,E}$ and $w_{h,m+1}$, $0 \le h \le t$, are dropped.

We have that $a^{S_{0,i}} = w_{0,i}$, $1 \le i \le m$, $a^{S_{0,E}} = w_{0,m+1}$.

If for any $h < t$, S_h and c_h satisfying the above relationship, i.e., $c_h = (a^{S_{h,1}}, \cdots, a^{S_{h,m}}, a^{S_{h,E}})$, then we show that $c_{h+1} = (a^{S_{h+1,1}}, \cdots, a^{S_{h+1,m}}, a^{S_{h+1,E}})$.

(i) First, we show that if $S_h = (S_{h,1}, \cdots, S_{h,m}, S_{h,E}) \Longrightarrow S_{h+1} = (S_{h+1,1}, \cdots, S_{h+1,m}, S_{h+1,E})$, then $c_{h+1} = (a^{S_{h+1,1}}, \cdots, a^{S_{h+1,m}}, a^{S_{h+1,E}})$. Indeed, in S_h for any $1 \le i \le m$ there is at most one rule $r_{j,i} \in R_i$, $1 \le j \le k_i$. In c_h the rules $r'_{j,i} \in R'_i$, $1 \le i \le m$, $1 \le j \le k_i$, are used in the next computation step in $k\Pi_e$. According to the definition of $k\Pi_e$ the rules $r'_{j,i}$ and their corresponding rules $r_{j,i}$ in Π_e are applied in the same circumstances (guards $g_{E_{j,i}}$ and regular expressions $E_{j,i}$ are satisfied for the same conditions), consume the same inputs and produce the same outputs. Hence, $c_{h+1} = (a^{S_{h+1,1}}, \cdots, a^{S_{h+1,m}}, a^{S_{h+1,E}})$.

(ii) The other part of the equivalence is similar.

It follows that Π_e and $k\Pi_e$ produce the same results and $\mathcal{N}(\Pi_e) = \mathcal{N}(k\Pi_e)$. □

In the sequel different variants of SN P systems are considered together with their mapping into kP systems. In each of these cases the focus will be on the new features introduced or changes made to standard or extended SN P systems and the ways these are reflected in the kP systems associated with them.

SN P systems with colored spikes [52] use a set of different types of spikes. Applications of this model have been studied in [33]. Formally, this is given by

$\Pi_{e,c} = (O, \sigma_1, \ldots, \sigma_m, syn, i_0)$, where the alphabet consists of *colored spikes*, $O = \{a_1, \cdots, a_g\}$, where $g \geq 1$, is the number of colors. For any neuron σ_i, $1 \leq i \leq m$, the rules of R_i have the form $r : E/a_1^{c_1} \cdots a_g^{c_g} \to a_1^{p_1} \cdots a_g^{p_g}$, where E is a regular expression over O, $c_j \geq p_j \geq 0$ and $c_j \geq 1$, $1 \leq j \leq g$. The result produced by $\Pi_{e,c}$ is the set of tuples $(x_1, \cdots, x_g)$, where x_j is the number of spikes of color $1 \leq j \leq g$, counted in the output neuron at the end of a halting computation or in the environment during a halting computation.

Corollary 2 *For any SN P system with colored spikes,* $\Pi_{e,c}$*, there exists a kP system,* $k\Pi_{e,c}$*, such that* $\mathcal{N}(\Pi_{e,c}) = \mathcal{N}(k\Pi_{e,c})$.

Proof Similar to the proof of Theorem 1, for a given SN P system with colored spikes, $\Pi_{e,c}$, one constructs a kP system, $k\Pi_{e,c}$. For any rule $r \in R_i$, $1 \leq i \leq m$, as above, a rule $r' : a_1^{c_1} \cdots a_g^{c_g} \to (a_1^{p_1} \cdots a_g^{p_g}, t_{i_1}) \cdots (a_1^{p_1} \cdots a_g^{p_g}, t_{i_{h_i}}) \; \{g_E\}$ is added to the set of rules R'_i.
The proof of equivalence of the two P systems is similar to the proof of Theorem 1.
□

SN P systems with multiple channels [46] use *channels* associated with synapses which distinguish certain synapses that are used for passing spikes between neurons; the rules of such P systems refer to channels instead of synapses. Each channel is described by a label from L, $L \subset \mathbb{N}$, i.e., (i, j, l), where (i, j) is a synapse linking the neurons σ_i and σ_j and l is the label of the channel associated to the synapse. The rules have the form $r : E/a^c \to a^p(l)$, $c \geq p \geq 0$, $c \geq 1$, $l \in L$. This means that when such a rule is applied in neuron σ_i the p spikes are sent to the neighbouring neurons connected with synapses of the same label, l, instead of all the neurons connected through synapses.

Corollary 3 *For any SN P system with multiple channels,* $\Pi_{e,mc}$*, there exists a kP system,* $k\Pi_{e,mc}$*, such that* $\mathcal{N}(\Pi_{e,mc}) = \mathcal{N}(k\Pi_{e,mc})$.

Proof The proof is almost identical with the proof of Teorem 1. The only difference consists in defining the rules: for each rule r in σ_i there exists a rule r' in the set of rules R'_i of the compartment type t_i which sends a^p objects only to the compartments corresponding to neighboring neurons linked with σ_i through channel l. □

An **SN P system with anti-spikes,** introduced in [28], has the following new features that do not exist in standard or extended SN P systems: (i) an alphabet consisting of two objects, a and $\bar{a}$, called *spikes* and *anti-spikes,* respectively and (ii) rules of the form $E/b^c \to b'$ or $E/b^c \to \lambda$, where each of b and b' are either a or $\bar{a}$. When both a and $\bar{a}$ appear in a neuron, after receiving them from other neurons, a rule $a\bar{a} \to \lambda$ is immediately applied in a maximal parallel manner, before any other rule is used, such that if one gets a^e and $\bar{a}^d$ then either a^{e-d} or $\bar{a}^{d-e}$, depending on whether $e \geq d$ or $d \geq e$, respectively, is obtained. This acts as an annihilation rule removing all pairs $a, \bar{a}$. When the environment is used for getting the results, then the output neuron sends only spikes and no anti-spikes. One can observe that the rules of these SN P systems are similar to those of the standard SN P systems using rules with no delay.

Theorem 4 *For any SN P system with anti-spikes,* $\Pi_{a\bar{a}}$*, there exists a kP system,* $k\Pi_{a\bar{a}}$*, such that* $\mathcal{N}(\Pi_{a\bar{a}}) = \mathcal{N}(k\Pi_{a\bar{a}})$*.*

Proof This proof will show how the two new features of SN P systems with anti-spikes are considered in their mapping into kP systems.

Let us consider an SN P system with anti-spikes, $\Pi_{a\bar{a}} = (O, \sigma_1, \ldots, \sigma_m, \mathit{syn}, i_0)$, where $O = \{a, \bar{a}\}$; $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, with $n_i \geq 0$, the initial number of spikes in the neuron, and R_i is the set of rules of type $E/b^c \to b'$ or $E/b^c \to \lambda$, where $b, b' \in O$; syn the set of synapses between neurons and i_0, the output neuron. We construct a kP system $k\Pi_{a\bar{a}} = (A, \mu, C_1, \cdots, C_n, i_0)$, similar to the proof of Theorem 1. The rules of $k\Pi_{a\bar{a}}$ are slightly different in this case: for every rule $r : E/b^c \to b' \in R_i$, where $b, b' \in \{a, \bar{a}\}$, a rule $r' : b^c \to (b', t_{i_1}) \ldots (b', t_{i_{h_i}})\ \{g_E\}$ is added to R'_i and when $r : E/b^c \to \lambda$ is in R_i, then a rule $r' : b^c \to \lambda\ \{g_E\}$ is added to R'_i. Every set of rules R'_i, $1 \leq i \leq m$, of C_i contains an additional rule, which is $r_e : a\bar{a} \to \lambda$, hence, the number of rules of R'_i is the number of rules of R_i plus one. The compartment C_i is constructed from the compartment type t_i with the execution strategy $\delta_i = \{r_e\}^T \{r'_1, \ldots, r'_{k_i}\}$, where $k_i \geq 1$, is the number of rules in R_i. The execution strategy is a sequence with two blocks, the first one is a maximal parallel execution of r_e and the second one is a choice from the set of rules associated to those in R_i.

One can easily observe that in any computation in $k\Pi_{a\bar{a}}$, in each step, apart from the first one, the first block of the execution strategy is equivalent to an annihilation rule and for the remaining objects a or $\bar{a}$, one of r'_j, $1 \leq j \leq k_i$, is selected to be executed. At the end of each step the components may contain both a and $\bar{a}$ and the annihilation is done in the next step. For this reason, if the result produced by $\Pi_{a\bar{a}}$ is obtained in the environment, then in the compartment C_{m+1} of $k\Pi_{a\bar{a}}$ there is no need for annihilation, as only objects a are sent to C_{m+1}. Otherwise, an additional step may be required for computing the result of $k\Pi_{a\bar{a}}$. Hence, if the number of steps of a halting computation of $\Pi_{a\bar{a}}$ is t then the corresponding halting computation of $k\Pi_{a\bar{a}}$ may require $t + 1$ steps.

In order to show that the two P systems produce the same results one can use the approach presented in the proof of Theorem 1. Consequently, $\mathcal{N}(\Pi_{a\bar{a}}) = \mathcal{N}(k\Pi_{a\bar{a}})$.

$\square$

Now, the case of **SN P systems using rules with delay** is considered. Similar to Theorem 1, the rules of these systems will be of the extended type, but will include a delay as well.

Theorem 5 *For any standard or extended SN P system using rules with delay,* $\Pi_{e,d}$*, there exists a kP system,* $k\Pi_{e,d}$*, such that either* $\mathcal{N}(k\Pi_{e,d}) = \mathcal{N}(\Pi_{e,d})$ *or* $\mathcal{N}(k\Pi_{e,d}) = \mathcal{N}(\Pi_{e,d}) + 3$*.*

Proof Let $\Pi_{e,d} = (O, \sigma_1, \ldots, \sigma_m, syn, i_0)$ be an SN P system, where $O = \{a\}$ is the alphabet; $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, is a neuron consisting of an initial number of spikes, $n_i \geq 0$, and a set of rules, R_i, of the form $E/a^c \to a^p; d$, where $d \geq 0$; syn is the set of synapses; and i_0 indicates the output neuron. If the set R_i, $1 \leq i \leq m$, has rules with delay and the total number of rules is k_i, $k_i \geq 1$, then the rules can be reordered such that the first rules are with no delay, $r_{j,i} : E_{j,i}/a^{c_{j,i}} \to a^{p_{j,i}}$, $1 \leq j \leq l_i < k_i$, $l_i \geq 0$, and the rest are with delay, $r_{j,i} : E_{j,i}/a^{c_{j,i}} \to a^{p_{j,i}}; d_{j,i}$, $l_i + 1 \leq j \leq k_i$. Let us denote by min the minimum of all l_i and Max the maximum of all k_i, $1 \leq i \leq m$. R_D, $R_D = Max - min$, denotes the maximum number of rules with delay that may appear in any of the neurons.

The following kP system, $k\Pi_{e,d} = (A, \mu, C_1, \cdots, C_n, i_0)$, is constructed, where

1. $A = \{a, b, f, o, c\} \cup \{a_j|\ min + 1 \leq j \leq Max\} \cup \{\nu_j|\ min + 1 \leq j \leq Max\}$ is the alphabet.
2. n, μ and i_0 are as in Theorem 1.
3. The compartment type is $t_i = (R'_i, \delta_i)$ and $w_{i,0} = a^{n_i}$ if R_i contains only rules with no delay and $w_{i,0} = a^{n_i} f^2 o$ if R_i contains rules with delay. The two additional symbols, f and o, are used to handle the simulation of the delay. If R_i contains only rules with no delay, then R'_i is built as in Theorem 1.
4. If R_i contains rules with delay and $p_{j,i} \geq 1$, then the set R'_i has the following rules $r'_{j,i} : a^{c_{j,i}} \to (a^{p_{j,i}}, t_{i_1}) \cdots\ (a^{p_{j,i}}, t_{i_{h_i}})\ \{g_{E_{j,i}} \wedge = o\}$, $1 \leq j \leq l_i$, $l_i \geq 0$ (corresponding to the rules with no delay, $r_{j,i}$, from σ_i); and, for the rules with delay, $r_{j,i}$, $l_i + 1 \leq j \leq k_i$, from σ_i,
   $r'_{j,i,1} : ofa^{c_{j,i}} \to c\nu_j a_j^{d_{j,i}}\ \{g_{E_{j,i}} \wedge = o\}$, $l_i + 1 \leq j \leq k_i$;
   $r'_{j,i,2} : c\nu_j \to fo(a^{p_{j,i}}, t_{i_1}) \cdots (a^{p_{j,i}}, t_{i_{h_i}})\ \{= a_j\}$, $l_i + 1 \leq j \leq k_i$;
   $r'_{j,i,3} : a_j \to \lambda\ \{\geq a_j\}$, $l_i + 1 \leq j \leq k_i$; and
   $r'_{k_i+1,i} : a \to \lambda\ \{(\geq a_{l_i+1}\ \vee \cdots \vee \geq a_{k_i})\}$;
   $r'_{k_i+2,i} : b \to a\ \{(= a_{l_i+1}\ \vee \cdots \vee = a_{k_i})\}$;
   $r'_{k_i+3,i} : a \to b\ \{= f\}$.
   The case $p_{j,i} = 0$, corresponding to a forgetting rule in $\Pi_{e,d}$, is handled similar to Theorem 1 and is left as an exercise.

   The case when the output neuron σ_{i_0} has no rules with delay is handled as in Theorem 1. If σ_{i_0} includes rules with delay, then the rules r'_{j,i_0}, $1 \leq j \leq l_{i_0}$, corresponding to rules with no delay in σ_{i_0}, are treated as in Theorem 1, whereas for each of the rules r'_{j,i_0}, $l_{i_0} + 1 \leq j \leq k_{i_0}$, corresponding to rules with delay in σ_{i_0}, the following changes are made: $(a^{p_{j,i_0}}, t_{m+1})$ is added to the right hand side of $r'_{j,i_0,2}$, which means that when σ_{i_0} will become open, the compartment C_{i_0} sends $a^{p_{j,i_0}}$ to C_{m+1}.
5. The execution strategy, δ_i, $1 \leq i \leq m$, is the sequence $B_1 B_2 B_3 B_4$, where
   $B_1 = \{r'_{1,i}, \cdots, r'_{l_i,i}\} \cup \{r'_{i,j,h} | l_i + 1 \leq j \leq k_i, 1 \leq h \leq 2\}$;
   $B_2 = \{r'_{k_i+1,i}, r'_{k_i+2,i}\}^T$; $B_3 = \{r'_{j,i,3}\}$; and $B_4 = \{r'_{k_i+3,i}\}^T$.

   B_1 and B_3 denote choice strategies, i.e., only one of the applicable rules from the sets defining the blocks is selected and executed. B_2 and B_4 blocks describe maximal parallel executions of the rules.

Subsequently, the kP system $k\Pi_{e,d}$ work is described.

In the compartments corresponding to neurons that do not contain rules with delay the rules are executed according to the choice execution strategy, as described in the proof of Theorem 1.

Now, we consider compartments C_i, $1 \leq i \leq m$, containing rules with delay and denote this case with (*). In each such compartment, when o is present, i.e, it corresponds to σ_i being open, a rule is applicable if its left hand side is contained in the current multiset of C_i and its guard is true. This happens in two cases, when: (i) the rule is $r'_{j,i}$, $1 \leq j \leq l_i$, or (ii) $r'_{j,i,1}$, $l_i + 1 \leq j \leq k_i$; both are from B_1. If $r'_{j,i}$, $1 \leq j \leq l_i$, is applied, then none of the rules from other blocks is applicable, due to their guards. Consequently, the execution of the corresponding rule, $r_{j,i}$, from R_i, with no delay, is simulated. If $r'_{j,i,1}$, $l_i + 1 \leq j \leq k_i$ is applied, then it starts simulating the behaviour of the rule $r_{j,i}$, with delay, from R_i. The rule $r'_{j,i,1}$ consumes objects o, f and $a^{c_{j,i}}$, producing c, ν_j and $a_j^{d_{j,i}}$ objects. The rule $r'_{j,i,1}$ simulates the first part of executing $r_{j,i}$, when the status of the neuron σ_i becomes closed (in C_i the object o is replaced by c and $a^{c_{j,i}}$, which represents the left hand side of $r_{j,i}$, is consumed). The $d_{j,i}$ objects a_j correspond to the delay $d_{j,i}$ introduced by $r_{j,i}$. After applying the rule, only one single f remains in C_i. Then, from the rules of the blocks B_2 to B_4, only $r'_{k_i+3,i}$ is executed in maximal parallel way, by transforming all the remaining objects a, if any, into objects b. Now, σ_i being closed, it won't send or receive any symbols for $d_{j,i} - 1$ steps. During that time, in C_i, the block B_1 is no longer executed, due to the guards of the rules being false. As long as $d_{j,i} \geq 1$, equivalent to the number of objects $a_j^{d_{j,i}}$ being greater than or equal to 1, the only executable rules are those occurring in blocks B_2 and B_3. The rule $r'_{k_i+1,i}$ from B_2, executed in maximal parallel manner, erases all the objects a that may have come from the compartments corresponding to neurons that send spikes to σ_i which is closed. The rule $r'_{j,i,3}$ from block B_3 decreases the number of a_j by one. The final block, B_4, is no longer executed as all the objects a have been consumed in block B_2. In the final step, when there is only one object a_j, the rules that are executed are $r'_{j,i,2}$, from B_1, and those from blocks B_2 and B_3. The rule $r'_{j,i,2}$ delivers $a^{p_{j,i}}$ objects to the linked compartments corresponding to the neighbours of σ_i, restoring the two objects f and introducing an object o corresponding to open status of σ_i. The rules $r'_{k_i+1,i}$ and $r'_{k_i+2,i}$, applied in a maximal parallel way, remove objects a previously received and restore the objects a that remained in C_i after applying $r'_{j,i,1}$, respectively. Finally, $r'_{j,i,3}$ removes the latest a_j. In this final step, $r'_{k_i+1,i}$ is not applicable as there ar no available objects a.

Let $N_{nd} = \{i | \sigma_i, 1 \leq i \leq m, \text{has only rules with no delay}\}$. If $\Pi_{e,d}$ uses the environment then $N_{nd} = N_{nd} \cup \{m+1\}$.

One can show, following Theorem 1 proof, that $S_0 \Longrightarrow S_1 \Longrightarrow \cdots \Longrightarrow S_t$ is a halting computation in $\Pi_{e,d}$, iff $c_0 \Longrightarrow c_1 \Longrightarrow \cdots \Longrightarrow c_t$, is a halting computation in $k\Pi_{e,d}$, with
$c_{h,i} = a^{S_{h,i}}$, when $i \in N_{nd}$,
$c_{h,i} = f^2 o a^{S_{h,i}}$, when $i \notin N_{nd}$ and σ_i at step h is open,
$c_{h,i}$ includes the objects $c f a_j^{d_{j,i}}$, when $i \notin N_{nd}$ and σ_i at step h is closed for $d_{j,i}$ steps; and $1 \leq h < t$.

From the case (*), it results that when σ_i is closed for $d_{j,i}$ steps, then $c_{h+d_{j,i},i} = f^2 o a^{S_{h+d_{j,i},i}}$ and σ_i is open at step $h + d_{j,i}$.

The results produced by the $k\Pi_{e,d}$ are the same with the results produced by $\Pi_{e,d}$, when the output neuron does not include rules with delay or when the results are obtained in the environment. Otherwise, if in the output neuron of $\Pi_{e,d}$ the result is the multiset M, then the result in the output compartment of $k\Pi_{e,d}$ is Mf^2o. Consequently, $\mathcal{N}(k\Pi_{e,d}) = \mathcal{N}(\Pi_{e,d})$, in the first case, and $\mathcal{N}(k\Pi_{e,d}) = \mathcal{N}(\Pi_{e,d}) + 3$, in the second case.

□

Example 2 A solution using SN P systems with delay to the logical gate introduced in Example 1 is presented in Fig 2. A rule with delay 1 appears in neuron σ_3. When the rule will spike an a, after a delay equal to 1, this is sent to σ_5 and σ_7. Other changes made to the SN P system presented in the Example 1 are the following: neuron σ_6 is now connected to σ_1 ($(1, 6) \in syn$) and σ_7 ($(6, 7) \in syn$), and the neuron σ_8 is no longer used.

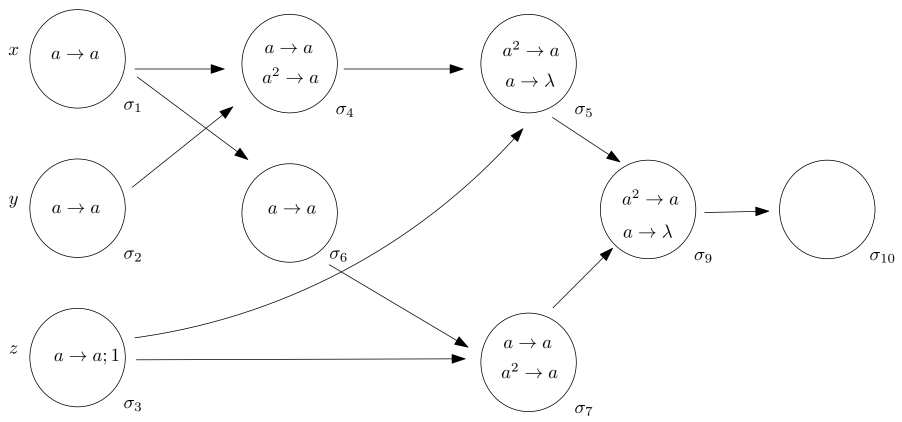

Fig. 2: SNP system with delay, $\Pi_{1,d}$, for $((x \vee y) \wedge z) \wedge (x \vee z)$

Table 3 shows how the SN P system with delay, $\Pi_{1,d}$, presented in Fig. 2, evolves from the start until the result is obtained for the input values $x = \lambda/a$, $y = z = a$.

Steps	$\sigma_1(x)$	$\sigma_2(y)$	$\sigma_3(z)$	σ_4	σ_5	σ_6	σ_7	σ_9	σ_{10}
0	λ/a	a	a	-	-	-	-	-	-
1	-	-	-	a/a^2	-	λ/a	-	-	-
2	-	-	-	-	a^2	-	a/a^2	-	-
3	-	-	-	-	-	-	-	a^2	-
4	-	-	-	-	-	-	-	-	a

Table 3: SN P system with delay, $\Pi_{1,d}$, sequence of steps

The kP system $k\Pi_{1,d} = (\{a, b, f, o, c, a_1, \nu_1\}, \mu, C_1, \cdots C_7, C_9, C_{10}, 10)$, built according to Theorem 5 is given by

$\mu = (\{C_1, \cdots C_7, C_9, C_{10}\}, V)$, and $V = \{\{C_1, C_4\}, \{C_1, C_6\}, \{C_2, C_4\}, \{C_3, C_5\}, \{C_3, C_7\}, \{C_4, C_5\}, \{C_5, C_9\}, \{C_6, C_7\}, \{C_7, C_9\}, \{C_9, C_{10}\}\}$; where each $C_i = (t_i, w_{i,0})$ is obtained from a compartment type, $t_i = (R'_i, \delta_i)$, $1 \le i \le 10$, $i \ne 8$. The initial multisets $w_{1,0} = x, w_{2,0} = y, w_{3,0} = zf^2o$, where each of the x, y, z is either a or λ, and $w_{i,0} = \lambda$, $4 \le i \le 10$, $i \ne 8$. The sets of rules are: $R'_1 = \{r'_{1,1} : a \to (a, t_4)(a, t_6)\}$, $R'_2 = \{r'_{1,2} : a \to (a, t_4)\}$, $R'_4 = \{r'_{1,4} : a \to (a, t_5)\ \{= a\}, r'_{2,4} : a^2 \to (a, t_5)\}$, $R'_5 = \{r'_{1,5} : a^2 \to (a, t_9), r'_{2,5} : a \to \lambda\ \{= a\}\}$, $R'_6 = \{r'_{1,6} : a \to (a, t_7)\}$, $R'_7 = \{r'_{1,7} : a \to (a, t_9)\ \{= a\}, r'_{2,7} : a^2 \to (a, t_9)\}$, $R'_9 = \{r'_{1,9} : a^2 \to (a, t_{10}), r'_{2,9} : a \to \lambda\ \{= a\}\}$, $R'_{10} = \emptyset$. Each execution strategy, $\delta_i, 1 \le i \le 9$, $i \ne 3, 8$, is choice and $\delta_{10} = \emptyset$.

R'_3 includes the following rules
$r'_{1,3,1} : ofa \to c\nu_1 a_1 \{= o\}$,
$r'_{1,3,2} : c\nu_1 \to fo(a, t_5)(a, t_7) \{= a_1\}$,
$r'_{1,3,3} : a_1 \to \lambda \{\geq a_1\}\}$,
$r'_{2,3} : a \to \lambda \{\geq a_1\}\}$,
$r'_{3,3} : b \to a \{= a_1\}\}$,
$r'_{4,3} : a \to b \{= f\}\}$.
One defines the following blocks: $B_1 = \{r'_{1,3,1}, r'_{i,3,2}\}$, $B_2 = \{r'_{2,3}, r'_{3,3}\}^T$, $B_3 = \{r'_{1,3,3}\}$ and $B_4 = \{r'_{4,3}\}^T$.

The execution strategy is the sequence $\delta_3 = B_1B_2B_3B_4$. Table 4 shows how the SN P system with delay, $k\Pi_{1,d}$, works. One can notice that f^2o is included in C_3 as additional objects and they remain in this neuron up until the end of the computation. In C_3 the rule $r'_{1,3,1}$ is applied to simulate the rule $r_{1,3}$ in σ_3; this rule consumes an object a and introduces $cf\nu_1 a_1$, showing that the compartment simulates the state close of σ_3 for one step. The object a is released to C_5 and C_7 in the third step, whereas the contents of C_2 become f^2o. In the other compartments there are only rules with no delay and $k\Pi_{1,d}$ behaves in these compartments similar to $k\Pi_1$. The two P systems produce the same results.

Steps	$C_1(x)$	$C_2(y)$	$C_3(z)$	C_4	C_5	C_6	C_7	C_9	C_{10}
0	λ/a	a	af^2o	-	-	-	-	-	-
1	-	-	$cf\nu_1 a_1$	a/a^2	-	λ/a	-	-	-
2	-	-	f^2o	-	a^2	-	a/a^2	-	-
3	-	-	f^2o	-	-	-	-	a^2	-
4	-	-	f^2o	-	-	-	-	-	a

Table 4: kP system with delay, $k\Pi_{1,d}$, sequence of steps

SN P systems with weights on synapses [32] use extended rules with delay, adding a new feature, positive integers, as *weights*, on synapses. For such an extended SN P system, one can write $syn \subseteq \{1, \dots, m\} \times \{1, \dots, m\} \times \mathbb{N}$. For a synapse (i, j, w), $i \neq j$, if p spikes are emitted by σ_i, then $p \times w$ spikes are received by σ_j.

Corollary 6 *For any SN P system with weights on synapses,* Π_w*, there exists a kP system,* $k\Pi_w$*, such that either* $\mathcal{N}(k\Pi_w) = \mathcal{N}(\Pi_w)$ *or* $\mathcal{N}(k\Pi_w) = \mathcal{N}(\Pi_w) + 3$.

Proof The kP system $k\Pi_w$ is obtained from the proof of Theorem 5, by adding the contribution of the weights to the right hand sides of $r'_{j,i}$ and $r'_{j,i,2}$.

The rules of the compartment type t_i, $1 \leq i \leq m$ have the following forms, when $p_{j,i} > 0$, $1 \leq j \leq k_i$, : $r'_{j,i} : a^{c_{j,i}} \to (a^{p_{j,i} \times w_1}, t_{i_1}) \cdots (a^{p_{j,i} \times w_{h_i}}, t_{i_{h_i}}) \{g_{E_{j,i}} \wedge = o\}$, where $1 \leq j \leq l_i$, $l_i \geq 0$ and $r'_{j,i,2} : c\nu_j \to fo(a^{p_{j,i} \times w_1}, t_{i_1}) \cdots (a^{p_{j,i} \times w_{h_i}}, t_{i_{h_i}}) \{= a_j\}$. The case $p_{j,i} = 0$, $1 \leq j \leq k_i$, can be easily handled from the proof of Theorem 5. □

A slightly different variant of SN P systems with weights is introduced in [55], where the concept of **SN P systems with weights, potential and threshold** are used. Rules with delay are not used for this model. The potential and threshold are computable real numbers (from the set $\mathbb{R}_c$). In such an SN P system of degree m, for each neuron, σ_i, $1 \leq i \leq m$, there is an initial potential, $p_i \in \mathbb{R}_c$, and a threshold, T_i, $T_i \geq 1$, attached to it. The neuron contains spiking rules of the form $T_i/q_{j,i} \to 1$, $1 \leq j \leq k_i$, for some $k_i \geq 1$ and $q_{j,i} \in \mathbb{R}_c$, such that $0 < q_{j,i} \leq T_i$. The threshold act as a guard of the rule, replacing the usual regular expression, and the potential appears instead of a spike. Similar to SN P systems with weights introduced in [32] and discussed above, every synapse has a weight, but now from a different set. The form of a synapse is (i, j, w), where $i, j \in \{1, \ldots, m\}$, $i \neq j$, and $w \in \mathbb{R}_c$ is the weight of the synapse. Given that at a moment, the potential of the neuron σ_i is $p \in \mathbb{R}_c$, a spiking rule is executed as follows. If $p = T_i$ then any rule belonging to the neuron σ_i, $T_i/q_{j,i} \to 1$, can be applied, decreasing the potential of σ_i by $q_{j,i}$ and sending a potential 1 to each of the neurons σ_j such that $(i, j, w) \in syn$. Neuron σ_j receives the potential w, which can be either positive or negative. One can note that a negative weight, w, acts similarly to an inhibitory synapse [28] that transforms spikes into anti-spikes (1, a positive potential, will arrive as w, a negative potential, in σ_j). When $p < T_i$ the potential will become 0, i.e., potential p will be removed from σ_i. If $p > T_i$ then no change of potential occurs in σ_i. In the proofs provided in [55], Turing completeness is obtained when $\mathbb{Z}$ is utilised instead of $\mathbb{R}_c$. The theorem below refers to weights in $\mathbb{Z}$.

Theorem 7 *For any SN P system with weights, potential and threshold, $\Pi_{w,p,t}$, there exists a kP system, $k\Pi_{w,p,t}$, such that $\mathcal{N}(\Pi_{w,p,t}) = \mathcal{N}(k\Pi_{w,p,t})$.*

Proof Let us consider an SN P system with weights, potential and threshold, $\Pi_{w,p,t} = (\sigma_1, \ldots, \sigma_m,\ syn, i_0)$, where $\sigma_i = (p_i, R_i)$, $1 \leq i \leq m$, with p_i the initial potential of the neuron, and R_i the set of rules of the form $T_i/q_{j,i} \to 1$, $1 \leq j \leq k_i$, such that $0 < q_{j,i} \leq T_i$. Similar to the proof of Theorem 1, a kP system, $k\Pi_{w,p,t} = (A, \mu, C_1, \cdots, C_n, i_0)$, is constructed. In this case we take $A = \{a, \bar{a}\}$ (here a and $\bar{a}$ play the roles of the objects used in the mapping of SN P systems with anti-spikes into kP systems).

The set of rules R'_i includes:

- $r'_{j,i} : a^{q_{j,i}} \to (b^{w_1}, t_{i_1}) \cdots (b^{w_{h_i}}, t_{i_{h_i}})\ \{= a^{T_i}\}$, where $b = a$, when $w_e > 0$ and $b = \bar{a}$, when $-w_e < 0$, $1 \leq e \leq h_i$, corresponding to $r_{j,i} : T_i/q_{j,i} \to 1$, $1 \leq j \leq k_i$;
- $r_{a,\bar{a}} : a\bar{a} \to \lambda\ \{\geq a \wedge \geq \bar{a}\}$, to simulate in C_i the difference between positive and negative potentials, when both appear in σ_i;
- $r_a : a \to \lambda\ \{< a^{T_i}\}$, corresponding to the case when the potential in σ_i is less than T_i; this will become 0;
- $r_{\bar{a}} : \bar{a} \to \lambda$, when a potential is negative then is always made 0.

The execution strategy is a sequence $\delta_i = \{r_{a,\bar{a}}\}^T \{r_a, r_{\bar{a}}\}^T \{r'_{1,i}, \cdots, r'_{k_i,i}\}$. If there are both positive and negative potentials, the difference of the two is made (first block executed in maximal parallel manner) and either the potential is less than T_i (second block of rules) or equal to T_i (last one, selecting arbitrarily one of the k_i rules) is executed.

Similar to the proof of Theorem 4, one can show that the results produced by the two P systems are the same and for any halting computation of $\Pi_{w,p,t}$ of t time steps the corresponding halting computation of $k\Pi_{w,p,t}$ has t or $t+1$ steps. $\square$

SN P systems with polarizations [38, 58] represent a variant of SN P systems whereby regular expressions are replaced by *electrical charges*, $+, -, 0$. Each neuron of the system has a specific electrical charge. The rules have the form $\alpha|a^c \rightarrow a^p;\beta$, where $\alpha, \beta \in \{+,-,0\}$, $c \geq 1$ and $0 \leq p \leq 1$. A rule is applicable when the number of spikes of the neuron σ_i, is greater than or equal to c and the electrical charge of the rule coincides with the electrical charge of the neuron. If the rule is applied, then c spikes are consumed from the neuron and a and β, when $p = 1$, or only β, when $p = 0$, are sent to all neighbouring neurons, σ_j, $(i, j) \in syn$. After electrical charges are received by a neuron, its current electrical charge plus those received follow the calculation: (i) several $+'$s, several $0'$s and several $-'$s produce one $+$, one 0 and one $-$, respectively; (ii) $+$ and $-$ lead to 0; (iii) $+$ or $-$ is not changed by 0.

Theorem 8 *For any SN P system with polarizations, Π_p, there exists a kP system, $k\Pi_p$, such that either $\mathcal{N}(k\Pi_p) = \mathcal{N}(\Pi_p)$ or $\mathcal{N}(k\Pi_p) = \mathcal{N}(\Pi_p) + 3$.*

Proof Let us consider an SN P system with polarizations, $\Pi_p = (O, \sigma_1, \ldots, \sigma_m,\ syn, i_0)$, where $O = \{a\}$, $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, with $n_i \geq 0$, the initial number of spikes in the neuron, and R_i the set of rules $\alpha|a^c \rightarrow a^p;\beta$, where $\alpha, \beta \in \{+,-,0\}$, $c \geq 1$ and $0 \leq p \leq 1$. The initial electrical charge of σ_i, $1 \leq i \leq m$, is α_i. syn is the set of synapses between neurons and i_0, indicates the output neuron. Let us consider the kP system $k\Pi_p = (A, \mu, C_1, \cdots, C_n, i_0)$. $A = \{a, +, -, 0, f\}$ is the alphabet. As in the previous cases, $n = m$ or $n = m + 1$. For each neuron σ_i, $1 \leq i \leq m$, with initial electrical charge α_i, a compartment $C_i = (t_i, w_{i,0})$ of type $t_i = (R_i, \delta_i)$ is constructed, with the initial multiset $w_{i,0} = a^{n_i} f^2 \alpha_i$, where a^{n_i} corresponds to the initial number of spikes of σ_i, α_i is the electrical charge of the neuron σ_i and f, occurring with multiplicity 2, is a flag used in computing the electrical charge from a multiset of electrical charges. For each set of rules R_i, with rules $r_{j,i}$, $1 \leq j \leq k_i$, one creates the set of rules R'_i. Before presenting the rules of R'_i, the following disjoint guards are defined
$g_0 = (\geq 0 \wedge \geq + \wedge \geq -) \vee (< 0 \wedge \geq + \wedge \geq -) \vee (\geq 0 \wedge < + \wedge < -)$;
$g_+ = (\geq 0 \wedge \geq + \wedge < -) \vee (< 0 \wedge\ \geq + \wedge < -)$;
$g_- = (\geq 0 \wedge \geq - \wedge < +) \vee (< 0 \wedge\ \geq - \wedge < +)$.
A guard g_x, $x \in \{+,-,0\}$, is true when the calculation based on steps (i) - (iii), given before Theorem 8, is applied to the multiset over $\{+,-,0\}$ from the current multiset of the compartment C_i and leads to x. For example, in the case of g_0 there are three possibilities when the occurrences of $+,-,0$ from C_i lead to 0: (a) at least one of $+,-,0$; (b) at least one of $+,-$ and no 0; and (c) at least one 0 and no $+,-$. For (a), applying first (i) then (ii) one gets two objects 0 and using again (i) yields 0. For (b), one $+$ and one $-$ are obtained by applying (i) and then one 0 is produced using (ii). Situation (c) requires only step (i) which leads to one 0.

The set of rules R'_i, $1 \leq i \leq m$, consists of the rules defined below.

- for every rule
  $r_{j,i} : \alpha_{j,i}|a^{c_{j,i}} \rightarrow y;\beta_{j,i} \in R_i$, $1 \leq j \leq k_i$, where $y = a$ or $y = \lambda$, R'_i contains
  $r'_{j,i} : a^{c_{j,i}} \rightarrow (y\beta_{j,i}, t_{i_1}), \cdots, (y\beta_{j,i}, t_{i_{h_i}})\ \{g_{\alpha_{j,i}}\}$, where $y = a$ or $y = \lambda$;

- $r'_x : f \rightarrow x \ \{g_x\}$, $x \in \{+, -, 0\}$;
- $r'_{x,e} : x \rightarrow \lambda \ \{= f\}$, $x \in \{+, -, 0\}$;
- $r'_f : f \rightarrow f^2 \ \{= f\}$.

The execution strategy $\delta_i = B_1 B_2 B_3$, is a sequence, where $B_1 = \{r'_{1,i}, \cdots, r'_{k_i,i}\}$, $B_2 = \{r'_+, r'_-, r'_0\}$, $B_3 = \{r'_{+,e}, r'_{-,e}, r'_{0,e}, r'_f\}^T$. B_1 is a choice, executing at most one of the rules $r'_{j,i}$, $1 \leq j \leq k_i$, with the guard $g_{\alpha_{j,i}}$ being true (σ_i and $r_{j,i}$ have the same electrical charge, $\alpha_{j,i}$). B_2 is also a choice, producing in C_i the object x when g_x is true. This corresponds to the electrical charge of σ_i based on the calculation steps mentioned above. Finally, all remaining objects, if any, corresponding to electrical charges are removed and the two objects f are restored, by executing the rules indicated by B_3 in a maximal parallel manner. It results that the object corresponding in a compartment C_i to the electrical charge of σ_i, $1 \leq i \leq m$, is obtained in the next step.

When the environment is used by the SN P system, an additional compartment is needed in $k\Pi_p$. The environment is used by Π_p only to collect the result, having no electrical charge. The corresponding compartment in $k\Pi_p$ may be defined as for the other kP systems, $C_{m+1} = (t_{m+1}, \lambda)$, where $t_{m+1} = (\emptyset, \emptyset)$. Also, a link $\{C_{i_0}, C_{m+1}\}$ is added to $k\Pi_p$ and the rule r'_{j,i_0} has on its right hand side only y instead of $y\beta_{j,i_0}$, which means that no electrical charge object is sent to C_{m+1}.

As in the proofs of the previous theorems, one can show that $S_0 \Longrightarrow S_1 \cdots \Longrightarrow S_t$ is a halting computation in Π_p, iff $c_0 \Longrightarrow c_1 \Longrightarrow \cdots \Longrightarrow c_{t'}$ is a halting computation in $k\Pi_p$, with $S_h = (S_{h,1}, \cdots, S_{h,m}), c_h = (c_{h,1}, \cdots, c_{h,m})$ and $c_{h,i} = S_{h,i} f^2 M_i$, $1 \leq h \leq t$, $1 \leq i \leq m$, where M_i is a multiset over $\{+, -, 0\}$; when $n = m + 1$, $c_{h,m+1} = S_{h,m+1}$, $1 \leq h \leq t$.

One can notice that if Π_p uses the environment and produces s spikes, then and $k\Pi_p$ produces the result a^s and $t' = t$. Otherwise, if Π_p produces the number of spikes s in the output neuron α_{i_0}, then $k\Pi_p$ produces in C_{i_0} the result $a^s f^2 \alpha_{i_0}$ and $t' = t + 1$. Hence, $\mathcal{N}(k\Pi_p) = \mathcal{N}(\Pi_p)$ or $\mathcal{N}(k\Pi_p) = \mathcal{N}(\Pi_p) + 3$.

□

Remark 2 If two rules from R'_i are identical, but the guards are different,
$r'_{j_1,i} : a^c \rightarrow (y\beta, t_{i_1}), \cdots, (y\beta, t_{i_{h_i}}) \ \{g_{\alpha_{j_1,i}}\}$, and
$r'_{j_2,i} : a^c \rightarrow (y\beta, t_{i_1}), \cdots, (y\beta, t_{i_{h_i}}) \ \{g_{\alpha_{j_2,i}}\}$, where $g_{\alpha_{j_2,i}} \neq g_{\alpha_{j_1,i}}$, $y = a$ or $y = \lambda$, then they can be replaced by a single rule with a guard being the disjunction of those that appear in the rules,
$r'_{j_1,i} : a^c \rightarrow (y\beta, t_{i_1}), \cdots, (y\beta, t_{i_{h_i}}) \ \{g_{\alpha_{j_1,i}} \vee g_{\alpha_{j_2,i}}\}$. One can apply the same process for three rules as well, where the disjunction of three conditions is used. No more than three such rules may appear as this is the maximum number of potential polarizations.

Example 3 In Figure 3 is presented a solution to the logical gate. This is formally given by: $\Pi_{1,p} = (\{a\}, \sigma_1, \cdots, \sigma_{10}, syn, 10)$, where the sets of rules are $R_1 = \{r_{1,1} : +|a \rightarrow a; +\}$, $R_2 = \{r_{1,2} : -|a \rightarrow a; -\}$, $R_3 = \{r_{1,3} : -|a \rightarrow a; -\}$, $R_4 = \{r_{1,4} : +|a \rightarrow a; +, r_{2,4} : -|a \rightarrow a; +, r_{3,4} : 0|a^2 \rightarrow a; +\}$, $R_5 = \{r_{1,5} : +|a \rightarrow a; +\}$, $R_6 = \{r_{1,6} : -|a \rightarrow a; -\}$, $R_7 = \{r_{1,7} : +|a \rightarrow \lambda; 0, r_{2,7} : -|a \rightarrow \lambda; 0, r_{3,7} : 0|a^2 \rightarrow a; +\}$, $R_8 = \{r_{1,8} : +|a \rightarrow a; -, r_{2,8} : -|a \rightarrow a; -, r_{3,8} : 0|a^2 \rightarrow a; -\}$, $R_9 = \{r_{1,9} : +|a \rightarrow \lambda; 0, r_{2,9} : -|a \rightarrow \lambda; 0, r_{3,9} : 0|a^2 \rightarrow a; 0\}$, $R_{10} = \emptyset$.

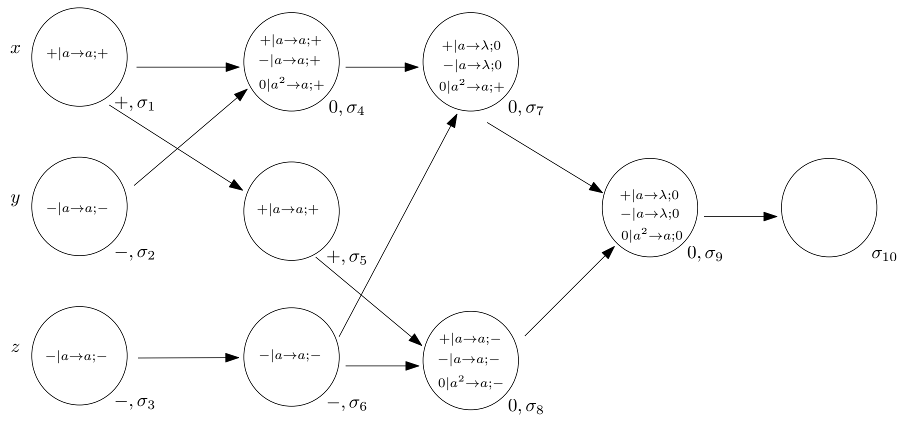

Fig. 3: SNP system with polarizations, $\Pi_{1,p}$, for $((x \vee y) \wedge z) \wedge (x \vee z)$

The Table 5 presents how this SN P system evolves. For each step one line is for the polarizations of the neurons and the second contains their contents, when the inputs $x = \lambda/a$, $y = x = a$ are used.

Steps	$\sigma_1(x)$	$\sigma_2(y)$	$\sigma_3(z)$	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}
0	$+$	$-$	$-$	0	$+$	$-$	0	0	0	0
	λ/a	a	a	λ	λ	λ	λ	λ	λ	λ
1	$+$	$-$	$-$	$-/0$	$+$	$-$	0	0	0	0
	λ	λ	λ	a/a^2	λ/a	a	λ	λ	λ	λ
2	$+$	$-$	$-$	$-/0$	$+$	$-$	0	$-/0$	0	0
	λ	λ	λ	λ	λ	λ	a^2	a/a^2	λ	λ
3	$+$	$-$	$-$	$-/0$	$+$	$-$	0	$-/0$	0	0
	λ	λ	λ	λ	λ	λ	λ	λ	a^2	λ
4	$+$	$-$	$-$	$-/0$	$+$	$-$	0	$-/0$	0	0
	λ	λ	λ	λ	λ	λ	λ	λ	λ	a

Table 5: SN P system with polarization, $\Pi_{1,p}$, sequence of steps

Now, one can build a kP system, $k\Pi_{1,p}$, using the construction provided in the proof of Theorem 8. The set of rules $\{r'_x : f \to x\ \{g_x\} | x \in \{+,-,0\}\} \cup \{r'_{x,e} : x \to \lambda\ \{= f\} | x \in \{+,-,0\}\} \cup \{r'_f : f \to f^2\ \{= f\}\}$, introduced in that proof, is denoted by R.

The kP system $k\Pi_{1,p} = (A, \mu, C_1, \cdots C_{10}, 10)$, is given by

1. $A = \{a, +, -, 0, f\}$;
2. $\mu = (\{C_1, \cdots C_{10}\}, V)$, and $V = \{\{C_1, C_4\}, \{C_1, C_5\}, \{C_2, C_4\}, \{C_3, C_6\}, \{C_4, C_7\}, \{C_5, C_8\}, \{C_6, C_7\}, \{C_6, C_8\}, \{C_7, C_9\}, \{C_8, C_9\}, \{C_9, C_{10}\}\}$;
3. each $C_i = (t_i, w_{i,0})$ is obtained from a compartment type, $t_i = (R'_i, \delta_i)$, $1 \le i \le 10$, with initial multisets $w_{1,0} = xf^2+, w_{2,0} = yf^2-, w_{3,0} = zf^2-$, where each of the x, y, z is either a or λ, and $w_{4,0} = w_{7,0} = w_{8,0} = w_{9,0} = f^2 0$, $w_{5,0} = f^2+$, $w_{6,0} = f^2-$, $w_{10,0} = f^2 0$. The sets of rules are $R'_1 = \{r'_{1,1} : a \to (a+, t_4)(a+, t_5)\ \{g_+\}\} \cup R$, $R'_2 = \{r'_{1,2} : a \to (a-, t_4)\ \{g_-\}\} \cup R$, $R'_3 = \{r'_{1,3} : a \to (a-, t_6)\ \{g_-\}\} \cup R$,

$R'_4 = \{r'_{1,4} : a \to (a+, t_7)\ \{g_+ \ \vee g_-\}, r'_{2,4} : a^2 \to (a+, t_7)\ \{g_0\}\} \cup R$, $R'_5 = \{r'_{1,5} : a \to (a+, t_8)\ \{g_+\}\} \cup R$, $R'_6 = \{r'_{1,6} : a \to (a-, t_7)(a-, t_8)\ \{g_-\}\} \cup R$, $R'_7 = \{r'_{1,7} : a \to (0, t_9)\ \{g_+ \ \vee g_-\}, r'_{2,7} : a^2 \to (a+, t_9)\ \{g_0\}\} \cup R$, $R'_8 = \{r'_{1,8} : a \to (a-, t_9)\ \{g_+ \ \vee g_-\}, r'_{2,8} : a^2 \to (a-, t_9)\ \{g_0\}\} \cup R$, $R'_9 = \{r'_{1,9} : a \to (0, t_{10})\ \{g_+ \ \vee g_-\}, r'_{2,9} : a^2 \to (a0, t_{10})\ \{g_0\}\} \cup R$, $R'_{10} = \emptyset$, and each execution strategy, $\delta_i, 1 \leq i \leq 9$, is a sequence defined with the three blocks introduced in the proof of Theorem 8 and $\delta_{10} = \emptyset$.

Table 6 presents the first four steps of executing $k\Pi_{1,p}$. Please note that every compartment in each step contains f^2, according to the definition of $k\Pi_{1,p}$, but this multiset is not present in the table in order to save space. For each step, including the initial one, 0, the first row contains the multisets of objects f^2 and those indicating the polarizations (for example, $f^2 0 - / f^2 0 + -$ for C_4 in step 1) and the second one points to the multisets over a (a/a^2, the same compartment and step). One can observe that the two tables illustrate the behaviour of the two P systems as presented in the proof of Theorem 8. The final result computed by $k\Pi_{1,p}$ requires one more step, when $f^2 0^2 a$ becomes $f^2 0a$ in C_{10}.

Steps	$C_1(x)$	$C_2(y)$	$C_3(z)$	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}
0	$+$	$-$	$-$	0	$+$	$-$	0	0	0	0
	λ/a	a	a	λ	λ	λ	λ	λ	λ	λ
1	$+$	$-$	$-$	$0-/0+-$	$+/+^2$	$-^2$	0	0	0	0
	λ	λ	λ	a/a^2	λ/a	a	λ	λ	λ	λ
2	$+$	$-$	$-$	$-/0$	$+$	$-$	$0+-$	$0-/0+-$	0	0
	λ	λ	λ	λ	λ	λ	a^2	a/a^2	λ	λ
3	$+$	$-$	$-$	$-/0$	$+$	$-$	0	$-/0$	$0+-$	0
	λ	λ	λ	λ	λ	λ	λ	λ	a^2	λ
4	$+$	$-$	$-$	$-/0$	$+$	$-$	0	$-/0$	0	0^2
	λ	λ	λ	λ	λ	λ	λ	λ	λ	a

Table 6: SN P system with polarization, $k\Pi_{1,p}$, sequence of steps

3.1.1 Complexity metrics

Below are compared the differences between the SN P systems and the corresponding kP systems resulted from the mappings presented so far. These are expressed through several *complexity metrics*. We start we some notations.

For a P system Π the following *complexity metrics* are considered

1. $Comp_\Pi$ = total number of Π's membranes (neurons for SN P systems, compartments for kP systems);
2. $Symb_\Pi$ = size of Π's alphabet (number of symbols);
3. $Rule_\Pi$ = total number of Π's rules;
4. *P system structure*: the graph associated with μ;
5. $Time_\Pi$ = number of time steps for computing a result.

The first three are called *descriptional complexity metrics.*

In the sequel are listed the values of these metrics for each pair of P systems presented in the mappings. We denote by m the number of neurons of an SN P system. These relationships are provided below, considering each of the SN P systems investigated.

Standard/extended SN P systems. From the proof of Theorem 1 one can extract the following results:

1. $Comp_{k\Pi_e} = Comp_{\Pi_e}$, when Π_e does not use the environment; otherwise $Comp_{k\Pi_e} = Comp_{\Pi_e} + 1$;
2. $Symb_{k\Pi_e} = Symb_{\Pi_e}$;
3. $Rule_{k\Pi_e} = Rule_{\Pi_e}$;
4. *P systems structure*: the graphs of $k\Pi_e$ and Π_e are the same (when Π_e uses the environment, then $k\Pi_e$ has an additional compartment);
5. $Time_{k\Pi_e} = Time_{\Pi_e}$.

The *execution strategy* in every compartment of $k\Pi_e$ is *choice*.

In most of the cases discussed in this paper, *Comp* and *P system structure* will be the same as above and they will be only mentioned when different.

SN P system with colored spikes and multiple channels. The metrics relationships and the execution strategies for these SN P systems are as in the previous case (Corollaries 2 and 3).

SN P systems with spikes and anti-spikes. (Theorem 4):

1. $Rule_{k\Pi_{a\bar{a}}} = Rule_{\Pi_{a\bar{a}}} + m$;
2. $Time_{k\Pi_{a\bar{a}}} = Time_{\Pi_{a\bar{a}}} + p$, where p is either 0 or 1.

The *execution strategy* of $k\Pi_{a\bar{a}}$ is a *sequence of two blocks*: the first is maximal parallelism and the second is a choice.

Standard/extended SN P systems with delay. (Theorem 5):

1. $Symb_{k\Pi_{e,d}} = Symb_{\Pi_{e,d}} + 2R_D + 4$, where $R_D = Max - min$;
2. $Rule_{k\Pi_{e,d}} \leq Rule_{\Pi_{e,d}} + (2R_D + 3)N_D$, where R_D is as above and N_D is the total number of neurons containing rules with delay.

The *execution strategy* of $k\Pi_{e,d}$ is a *sequence of four blocks*: choice, maximal parallelism, choice, maximal parallelism, in this order.

SN P systems with weights on synapses. The same relationships and execution strategy as in the case of standard/extended SN P systems with delay.

SN P systems with weights, potential and threshold. (Theorem 7):

1. $Rule_{k\Pi_{w,p,t}} = Rule_{\Pi_{w,p,t}} + 3m$.

The *execution strategy* of $k\Pi_{w,p,t}$ is a *sequence of three blocks*: the first two are maximal parallelism and the last is choice.

SN P systems with polarizations. (Theorem 8):

1. $Symb_{k\Pi_p} = Symb_{\Pi_p} + 4$;

2. $Rule_{k\Pi_p} = Rule_{\Pi_p} + 7m$;
3. $Time_{k\Pi_p} = Time_{\Pi_p} + p$, p is 0 or 1.

The *execution strategy* of $k\Pi_p$ is a *sequence of three blocks*: the first two are choices and the last is maximal parallelism.

3.2 Other SN P system topics and connections with kP systems

In this section are discussed: a different approach on obtaining the result of a computation and other execution strategies for SN P systems, in the context of mapping these systems into kP systems.

3.2.1 A different way of computing the result produced by an SN P system

In Subsection 2.1 is mentioned another way of computing the result of an SN P system, which consists in the number of steps executed by the SN P system between the first two spikes sent out to the environment by the output neuron - this is called type (ii) mode of computing the result. We focus on standard and extended SN P systems with no delay.

Theorem 9 *For any standard/extended SN P system, $\Pi_{e,ii}$, using type (ii) mode of computing the result, there exists a kP system, $k\Pi_{e,ii}$, such that $\mathcal{N}(k\Pi_{e,ii}) = \mathcal{N}(\Pi_{e,ii})$.*

Proof We start with an extended SN P system with no delay, $\Pi_{e,ii} = (O, \sigma_1, \cdots, \sigma_m, syn, i_o)$, which requires the use of the environment.

The following kP system, $k\Pi_{e,ii} = (A, \mu, C_1, \cdots, C_{n'}, n')$, where $n' = m + 2$, is constructed. The alphabet is $A = \{a, s, c, f\}$. Two new compartments are considered: C_{m+1} given by $C_{m+1} = (t_{m+1}, \lambda)$ and $t_{m+1} = (\emptyset, \emptyset)$, associated with the environment of the SN P system and C_{m+2} given by $C_{m+2} = (t_{m+2}, \lambda)$ and $t_{m+2} = (\emptyset, \emptyset)$, introduced for collecting the result of the computation, which reflects the number of steps executed by $\Pi_{e,ii}$ between the time steps when the first two spike are sent out to environment.

For every neuron σ_i, different from the output neuron, $i \neq i_0$, the component C_i has the initial multiset and the set of rules as in the proof of Theorem 1.

In C_{i_0} the initial multiset is $w_{i_0,0} = a^{n_{i_0}} s^4 c$ and for any rule $r_{j,i_0} : E_{j,i_0}/a^{c_{j,i_0}} \to b$, $b \in \{a, \lambda\}$, $1 \leq j \leq k_{i_0}$, the following rule is added to R'_{i_0}, $r'_{j,i_0} : sa^{c_{j,i_0}} \to (a, t_{m+1}) \; \{g_{E_{j,i_0}}\}$, when $b = a$ in r_{j,i_0}, $1 \leq j \leq k_{i_0}$; or $r'_{j,i_0} : a^{c_{j,i_0}} \to \lambda \; \{g_{E_{j,i_0}}\}$, when $b = \lambda$ in r_{j,i_0}, $1 \leq j \leq k_{i_0}$.

The object s is used for counting the number of steps executed between the times when the two rules corresponding to those from σ_{i_0} send the first and the second spike, respectively, to the environment. The symbol c is present in C_{i_0} only until the first rule mentioned before is executed.

The following rules will be also added to R'_{i_0}:

$r'_{k_{i_0}+1,i_0} : cs \to c \; \{= s^3\}$,
$r'_{k_{i_0}+2,i_0} : cs \to cs(s, t_{m+2}) \; \{(= s^2 \vee = s) \wedge < f\}$,
$r'_{k_{i_0}+3,i_0} : s \to sf \; \{= s \wedge < f\}$,
$r'_{k_{i_0}+4,i_0} : f \to sf \; \{< s \wedge = f\}$.

The execution strategy is δ_{i_0}, defined as a sequence of the blocks of rules B_1, B_2 where $B_1 = \{r'_{j,i_0} | 1 \leq j \leq k_{i_0}\}$ denotes a choice and $B_2 = \{r'_{k_{i_0+j},i_0} | 1 \leq j \leq 4\}^T$ specifies maximal parallel execution of the rules.

Given the guards of the rules from B_2, none of them is executed until a rule r'_{j,i_0} sending an a to C_{m+1} is executed. When such a rule is applied (it corresponds to sending out the first spike to the environment by the corresponding rule of σ_{i_0}), then the rule $r'_{k_{i_0+1},i_0}$ is executed, as there are three objects s left in C_{i_0}, and these are reduced to two by executing the rule. Since that moment onward, if for $k, k \geq 0$, time steps none of the rules r'_{j,i_0} that send an a to C_{m+1} is executed, then only the rule $r'_{k_{i_0+2},i_0}$ from B_2 is applied, which sends an s to C_{m+2}, the compartment collecting the result (equal to the number of time steps between the times when the rules corresponding to those in σ_{i_0} sending the first two spikes to the environment are executed). When a rule r'_{j,i_0} sending an a to C_{m+1} is selected to be applied, then a single s is left in C_{i_0} and $r'_{k_{i_0+2},i_0}$, $r'_{k_{i_0+3},i_0}$, from B_2, are both executed, sending the final s to C_{m+2} and introducing an f in C_{i_0}, respectively. After that moment, if r'_{j,i_0}, with λ on the right side is selected to be applied or no rule at all is used from B_1, then no rule from B_2 is executed. If r'_{j,i_0}, sending an a to C_{m+1}, is selected to be applied, then $r'_{k_{i_0+4},i_0}$ is executed, which restores the object s consumed by the previous rule. Hence, at the end of the computation, in C_{m+2} is obtained the number of objects s equal to the number of time steps executed by $\Pi_{e,ii}$ between the first two spikes sent out from σ_{i_0}. In C_{m+1} are collected exactly the number of objects a equal to the number of spikes sent to the environment by σ_{i_0}.
$\square$

There is an interesting aspect related to the *execution strategies* which *are not the same across the kP system*: in each of the compartments C_i, $1 \leq i \leq m$, $i \neq i_0$, the execution strategy is *choice*; in C_{i_0} this is a *sequence of two blocks.*

3.2.2 Other strategies of using the rules for SN P systems

So far, all the SN P systems presented have used the same strategy of using the rules, maximal parallelism at the system level and at most one rule per neuron in every step. In contrast to this, the kP systems presented so far have shown that their execution strategies may vary depending on the type of SN P systems considered. Subsequently, four other strategies of applying the rules are considered for SN P systems [22]. One focuses on standard and extended SN P systems with no delay, showing how these execution strategies are represented in the corresponding kP system.

The execution strategies considered are: *maximal parallelism mode* (denoted *mp*) – in each neuron the maximal parallelism of tissue-like P systems is applied; *exhaustive mode* (denoted *ex*) and *generalized use of rules* (denoted *g*) – when in each neuron at most one rule is selected, amongst those which are applicable, and for *ex* mode the rule is executed as many times as possible (maximal execution of a rule) and for *g* mode it is executed an arbitrary number of times; and *flat maximal parallelism mode* (denoted *fmp*) – when in each neuron a set of applicable rules is executed if it is maximal with respect to the inclusion order.

Theorem 10 *For any standard/extended SN P system, $\Pi_{e,\alpha}$, using the execution strategy $\alpha \in \{mp, ex, g, fmp\}$, there exists a kP system, $k\Pi_{e,\alpha}$, such that $\mathcal{N}(k\Pi_{e,\alpha}) = \mathcal{N}(\Pi_{e,\alpha})$.*

Proof For the standard/extended SN P system $\Pi_{e,mp}$ using maximal parallelism, we consider the kP system built in the proof of Theorem 1 and change the execution strategy in each compartment from *choice* to *maximal parallelism*, and denote it with $k\Pi_{e,mp}$. Obviously, $\Pi_{e,mp}$ and $k\Pi_{e,mp}$ compute the same result.

For $\alpha \in \{ex, g\}$ we consider a standard/extended SN P system, $\Pi_{e,\alpha} = (O, \sigma_1, \cdots, \sigma_m, syn, i_o)$, as in the previous proofs, and build the kP system $k\Pi' = (A, \mu, C_1, \cdots, C_n, i_0)$, where $n = m$ or $n = m + 1$; $A = \{a, x\} \cup \{f_i | 1 \le i \le K\}$, where $K = max\{k_i | 1 \le i \le m\}$; μ is as in the proof of Theorem 1; C_i, corresponding to σ_i, are instantiated from types t_i, $1 \le i \le m$, and the initial multiset of C_i is $w_{i,0} = a^{n_i} x f_1 \cdots f_K$. The set of rules R'_i of t_i has the following rules:
for each rule $r_{j,i} : E_{j,i}/a^{c_{j,i}} \to a^{p_{j,i}} \in R_i$, $1 \le j \le k_i$, of the neuron σ_i, $1 \le i \le m$, we have
$r'_{j,i,1} : f_j \to \lambda \ \{g_{E_{j,i}}\}$, $1 \le j \le k_i$;
$r'_{j,i,2} : a^{c_{j,i}} \to (a^{p_{j,i}}, t_{i_1}) \cdots \ (a^{p_{j,i}}, t_{i_{h_i}}) \ \{< f_j\} \ 1 \le j \le k_i$;
$r'_{j,i,3} : x \to x f_j \ \{< f_j\}$, $1 \le j \le k_i$.

When $\alpha = ex$ we consider the kP system $k\Pi_{e,ex}$ obtained from $k\Pi'$ where the execution strategy of t_i, $1 \le i \le m$, is given by $\delta_i = \{r'_{1,i,1}, \cdots, r'_{k_i,i,1}\} \ \{r'_{1,i,2}, \cdots, r'_{k_i,i,2}\}^T$ $\{r'_{1,i,3}, \cdots, r'_{k_i,i,3}\}$. This is a sequence of three blocks: the first block, a choice, selects and runs a rule $r'_{j,i,1}$ with its guard true, consuming f_j; the second block executes in a maximal manner the rule $r'_{j,i,2}$ given that $r'_{j,i,1}$ has been previously executed (this corresponds to the maximal execution of $r_{j,i}$ in σ_i); finally, the rule $r'_{j,i,3}$ from the last block is executed restoring f_j.

When $\alpha = g$ we consider the kP system $k\Pi_{e,g}$ constructed similar to the previous case, where in the execution strategy the second block is replaced by $\{r'_{1,i,2}, \cdots, r'_{k_i,i,2}\}^*$, which executes the selected rule an arbitrary number of times.

Obviously, in each of these two cases the SN P system and the corresponding kP system compute the same result.

For $\alpha = fmp$ we consider a standard/extended SN P system, $\Pi_{e,fmp} = (O, \sigma_1, \cdots, \sigma_m, syn, i_o)$, and construct a kP system, $k\Pi_e$ as in the proof of Theorem 1, where for each rule $r_{j,i} : E_{j,i}/a^{c_{j,i}} \to a^{p_{j,i}} \in R_i$, $1 \le j \le k_i$, with $c \ge p \ge 0$, $c \ge 1$, in neuron σ_i, $1 \le i \le m$, one have $r'_{j,i} : a^{c_{j,i}} \to (a^{p_{j,i}}, t_{i_1}) \cdots (a^{p_{j,i}}, t_{i_{h_i}}) \ \{g_{E_{j,i}}\}$, in R'_i of t_i.

For any t_i, $1 \le i \le m$, and q, $1 \le q \le k_i$, we denote by J_q the q-tuples of indexes of rules from R'_i, i.e., $J_q = \{(j_1, \cdots, j_q) | 1 \le j_e \le k_i, 1 \le e \le q, j_1 \ne \cdots \ne j_q\}$, and consider for each $(j_1, \cdots, j_q) \in J_q$ the rules $r'_{j_e,i}$, $1 \le e \le q$. One denotes by $a^{c_{j_1 \cdots j_q}}$ the concatenation of the left side of the rules; $a^{p_{j_1 \cdots j_q}}$ is obtained similarly by concatenating their right side (one can observe that not always $a^{p_{j_1 \cdots j_q}}$ is the concatenation of a^{j_e}, $1 \le e \le q$) and considers the rule $r''_{j_1 \cdots j_q} : a^{c_{j_1 \cdots j_q}} \to f a^{p_{j_1 \cdots j_q}} \ \{g_{E_{j_1,i}} \wedge \cdots \wedge g_{E_{j_q,i}} \wedge < f\}$, where f is a new symbol. One denotes by B_q the set $\{r''_{j_1 \cdots j_q} | (j_1, \cdots, j_q) \in J_q\}$ and define $R''_i = B_1 \cup \cdots \cup B_{k_i} \cup \{r''_f : f \to \lambda\}$, where $r''_f : f \to \lambda$ is a new rule.

Now, we consider the kP system $k\Pi_{e,fmp}$, obtained from $k\Pi_e$, where the alphabet of $k\Pi_{e,fmp}$ is $A = \{a, f\}$; μ is the same for the two kP systems; each type t'_i, $1 \le i \le m$, is given by the sets R''_i and the execution strategy $\delta'_i = B_{k_i} \& \ldots \& B_1 \& \{r''_f\}$, where each block of the sequence is a choice. Each compartment C'_i of type t'_i has the same initial multiset as C_i.

Now, one can show that the kP system $k\Pi_{e,fmp}$ produces the same results as $\Pi_{e,fmp}$. Indeed, the flat maximal parallelism of Π in each neuron σ_i is simulated by the execution strategy δ'_i defined above. This is a sequence of $k_i + 1$ blocks, each of them being a choice. One notices that B_{k_i}, the first block of the execution strategy, corresponds to the execution of all the rules, if at all possible, and B_1 executes one of the rules of R'_i. In fact, the set B_1 is equal to R'_i. This sequence checks which is the largest set of R'_i that can be executed and runs it. Whenever in a block B_q a rule is selected to be executed, an f is also introduced and the rest of the blocks, $B_{q-1}, \dots, B_1$ are no longer applicable and finally f is removed, by executing r''_f.

$\square$

3.2.3 Other complexity metrics for standard/extended SN P systems

Standard/extended SN P systems computing the result as the number of steps between the first two spikes. (Theorem 9):

1. $Comp_{k\Pi_{e,ii}} = Comp_{\Pi_{e,ii}} + 2$;
2. $Symb_{k\Pi_{e,ii}} = Symb_{\Pi_{e,ii}} + 3$;
3. $Rule_{k\Pi_{e,ii}} = Rule_{\Pi_{e,ii}} + 4$;
4. *System structure:* one more compartment added to $k\Pi_{e,ii}$.

The *execution strategy* of every compartment of $k\Pi_{e,ii}$ different from the output compartment is *choice* and for the output compartment is a *sequence of two blocks*: choice followed by maximal parallelism.

Standard/extended SN P systems. (Theorem 10):
when $\alpha \in \{ex, g\}$:

1. $Symb_{k\Pi_{e,\alpha}} = Symb_{\Pi_{e,\alpha}} + K + 1$;
2. $Rule_{k\Pi_{e,\alpha}} = 3Rule_{\Pi_{e,\alpha}}$.

when $\alpha = fmp$:

1. $Symb_{k\Pi_{e,fmp}} = Symb_{\Pi_{e,fmp}} + 1$;
2. $Rule_{k\Pi_{e,fmp}} = 2^{k_1} + \cdots + 2^{k_m} - m$.

The *execution strategy* in the compartments of $k\Pi_{e,\alpha}$ is: *maximal parallelism*, when $\alpha = mp$; *sequence of three blocks*: choice, maximal parallelism, choice, when $\alpha = ex$; *sequence of three blocks*: choice, arbitrary execution, choice, when $\alpha = g$. If $\alpha = fmp$, then in each compartment, C_i, $1 \le i \le m$, the execution strategy is a *sequence of* $k_i + 1$ *blocks*, each of them *choice*, where k_i is the number of rules of σ_i.

4 Testing and verification

Membrane computing models have been developed for a large spectrum of applications [59, 6]. Their functionality should be validated and checked for errors or unintended behaviour. This can be obtained through various methods and techniques. We show below, using Example 1, how model based testing and formal verification using model checking can be used in this respect, pointing also to the complementarity of these approaches.

4.1 Model based testing

Model based testing methods for applications using membrane computing models have been considered, especially those derived from finite state machine based testing [15]. Special classes of membrane systems used for testing purposes have been considered and studied in relation to X-machines, which are extensions of finite state machines [10]. Testing methods for spiking neural P systems [16] and kernel P systems [17] have been introduced in the context of learning X-machines associated with them. A different type of testing techniques, using search based software engineering approaches, have been introduced for applications using basic P systems [53].

In the sequel we illustrate such a testing method for an application based on Example 1. One uses a learning algorithm for kernel P systems based on learning X-machines that are equivalent to these P systems for computations of bounded length. The inferred X-machine is used to generate test sets [17]. Such a test set that covers all the states of the X-machine is used to reveal the errors of the application mentioned above. Errors seeded into the application using mutants [21] will be used to illustrate the efficacy of the test set.

For the SN P system Π_1 introduced in Example 1 the kP system $k\Pi_1$ is obtained using Theorem 1. We use the kP system $k\Pi_1$ instead of the original model, Π_1, because for the former we have a tool, kPWorkbench [1], allowing to simulate the kP system and then apply the learning algorithm to get the X-machine that generates the test set.The two P system models, simulations of the kP system, the learning algorithm, tests sequences and errors discovered by these sequences are available from the repository [27]. The learning algorithm inferring an X-machine [17] is applied for an upper bound limit equal to 4 for $k\Pi_1$ computations. The test set generated by the learning algorithm covers all the states of the X-machine corresponding to all the configurations of the kP system that are reachable in maximum 4 steps. Each test sequence includes the labels of the rules applied in the above mentioned calculations and the starting and ending configurations in each computation step. The test sequences obtained will n be applied to the implementation of the logical gate presented in Example 1.

The SN P system, Π_1, and the corresponding kP system, $k\Pi_1$, contain ten neurons (compartments) each. It follows that a configuration will have ten multiset values, generically denoted $[m_1, m_2, \cdots, m_{10}]$. In each computation step at most one rule is used in each neuron of Π_1 (compartment of $k\Pi_1$). The notations used in Π_1 and $k\Pi_1$ for rules will be used subsequently in describing test sequences. Some of the test sequences are listed below, showing the rules and configurations involved. We consider the input spikes a, λ and a for the input neurons σ_1, σ_2 and σ_3, respectively. These are the input values corresponding to $x = 1, y = 0$ and $z = 1$. The initial configuration is $[a, \lambda, a, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda]$.

Below are listed some test sequences generated based on the learning algorithm included in kPWorkbench. They start from the initial configuration, with
length = 1: $[a, \lambda, a, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda]$ $/\{r_{1,1}, r_{3,1}\}/$ $[\lambda, \lambda, \lambda, a, \lambda, a, a^2, \lambda, \lambda, \lambda]$;
length = 2: $[a, \lambda, a, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda]$ $/\{r_{1,1}, r_{3,1}\}/$ $[\lambda, \lambda, \lambda, a, \lambda, a, a^2, \lambda, \lambda, \lambda]$ $/\{r_{4,1}, r_{6,1}, r_{7,2}\}/$ $[\lambda, \lambda, \lambda, \lambda, a^2, \lambda, \lambda, a, \lambda, \lambda]$;
and length = 4: $[a, \lambda, a, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda]$ $/\{r_{1,1}, r_{3,1}\}/$ $[\lambda, \lambda, \lambda, a, \lambda, a, a^2, \lambda, \lambda, \lambda]$

$/\{r_{4,1}, r_{6,1}, r_{7,2}\}/\ [\lambda, \lambda, \lambda, \lambda, a^2, \lambda, \lambda, a, \lambda, \lambda]\ /\{r_{1,1}, r_{3,1}\}/$
$[\lambda, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda, a, \lambda]\ /\{r_{9,1}\}/\ [\lambda, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda, \lambda, a]$.

One can notice that each of the shorter sequences is a prefix of a larger one. Hence, the configurations covered by a sequence of length 1 are also covered by those of lengths 2 and 4, and the sequence of length 4 includes the configurations of any shorter sequences that are its prefixes. For this reason, only test sequences of length 4 will be used below.

We consider now mutants applied to the implementation of the the SN P system Π_1. The mutants used will be generated by applying the following mutation operations: a spiking rule is replaced by a forgetting rule, i.e., $a^s \to a$ becomes $a^s \to \lambda$. It can be observed that each of them will be discovered by a test sequence revealing the configuration that is corrupted and pointing to the error. This approach is similar to the experiments made in [21] for a broader set of mutants. The test sequence must be one of those where the mutated rule is present. For example, if the rule $r_{1,1} : a \to a$ is mutated as $r_{1,1}^{(m)} : a \to \lambda$ then any of the above listed test sequences will reveal this error as the second configuration is now $[\lambda, \lambda, \lambda, \lambda, \lambda, a, a, \lambda, \lambda, \lambda]$ which is different from the expected one $[\lambda, \lambda, \lambda, a, \lambda, a, a^2, \lambda, \lambda, \lambda]$, showing that an error in using $r_{1,1}$ did happen. Similarly, by mutating $r_{4,1} : a \to a$ into $r_{4,1}^{(m)} : a \to \lambda$ one gets a similar corrupted configuration revealing the error. If $r_{9,1} : a^2 \to a$ is mutated into $r_{9,1}^{(m)} : a^2 \to \lambda$ then this error is also identified by the same testing sequence. Obviously, shorter teste sequences would have revealed the first two errors.

All the test sequences, some mutation testing, the SN P system and kP model, as well as the X-machine inferred model are available at [27].

We claim that for this example the test sequences generated by the method presented above will reveal all the errors produced by mutants obtained by changing either the left hand side or right hand side of any rule in Π_1.

4.2 Verification

Formal verification is a method used to ensure that a model is correct before it is used to develop a system. *Model checking* is a type of formal verification in which the behaviour of a system, specified as a formal model, is analysed against various properties or queries expressed in a specific temporal logic.

Kernel P systems can be specified in a domain specific language as part of the software framework, called kPWorkbench [1, 20], which also includes a verification component [11] allowing to map a kP system into various model checkers. One of these model checkers used in connection with kP systems verification is NuSMV [11]. The modelling, simulation and verification aspects of this software framework have been presented in [7, 8]. For the example mentioned above, we present two sets of properties which are listed in Table 7. Each property is defined in three ways. Firstly, this is formulated using natural language-like patterns and then two translations into LTL and CTL logics are provided.

The properties below are verified for the kP system $k\Pi_1$ corresponding to the SN P system Π_1 presented in Example 1. All these properties can be expressed for Π_1 by replacing compartment C_i by neuron σ_i, objects by spikes and they remain true in this context. In the presentation below we will refer to some properties of the kP system $k\Pi_1$ connecting them to test sequences used for testing the application based on the SN P system Π_1. In fact, these properties should refer to Π_1, as we aim to validate Π_1, by using $k\Pi_1$ as an analysis instrument.

The first eight properties specify that for each of the possible input values, x, y, z the expected result is obtained. Ni.a gives the number of a's in compartment C_i, $1 \leq i \leq 3$, one of the input compartments of the kP system. N10.a refers to the number of a's in the compartment C_{10}, where the result of the logical gate is obtained.

The result of the logical gate is true for the following inputs: $x = 0, y = 1, z = 1$ or $x = 1, y = 0, z = 1$ or $x = 1, y = 1, z = 1$. These are expressed by the temporal logic properties defined in cases 4, 6 and 8, respectively, which all return true. The other cases, returning false, correspond to the situations when the inputs to the logical gate lead to false results. The sixth case corresponds to the testing sequence of length 4 discussed above. For each of the eight properties there is a test sequence of length less than or equal to 4. When there is no input, i.e., case 1 in the table, then the corresponding test sequence (of length 0) is λ. When only $y = 1$ and the rest are 0, leading to a false result, i.e., case 3 in the table, then the corresponding test sequence of length 3 is $/r_{2,1}/r_{4,1}/r_{5,2}/$ leading to a result 0 in C_{10}. For the other six cases there are sequences of length 4, three leading to 1, cases 4, 6 and 8, and three to 0, cases 2, 5 and 7. All these can be found in the test set available at the link provided.

The other two properties express some invariants. The ninth property proves that if at least one of the variables, x, y, z is 1 then C_5 or C_9 contains at least an a. The tenth property considers only two entries x, z and proves that if at least one is 1, then irrespective of the value of y, C_8 contains an a. Each of these properties makes correlations between different inputs and some intermediary values of the model and cannot be replicated by any single test sequence. The test sequences provided, in contrast to the properties listed in Table 7, offer more details, such as all the intermediary steps, with rules and configurations. These facts outline that the two approaches provide some similar results, but also refer to distinct useful insights that help analysing the given system from various perspectives.

The two validation methods presented above make effective use of the investigation on mapping different classes of SN P systems into kP systems.

	Property	
1	`always ((N1.a = 0  and  (N2.a = 0  and  N3.a = 0))  implies (eventually  N10.a = 0));`	false
	`LTLSPEC G (((N1.a = 0 & (N2.a = 0 & N3.a = 0)) -> F (N10.a = 0 & pInS)) | !pInS)`	
	`SPEC AG (((N1.a = 0 & (N2.a = 0 & N3.a = 0)) -> EF (N10.a = 0 & pInS)) | !pInS)`	
2	`always ((N1.a = 0  and  (N2.a = 0  and  N3.a = 1))  implies (eventually  N10.a = 0));`	false
	`LTLSPEC G (((N1.a = 0 & (N2.a = 0 & N3.a = 1)) -> F (N10.a = 0 & pInS)) | !pInS)`	
	`SPEC AG (((N1.a = 0 & (N2.a = 0 & N3.a = 1)) -> EF (N10.a = 0 & pInS)) | !pInS)`	
3	`always ((N1.a = 0  and  (N2.a = 1  and  N3.a = 0))  implies (eventually  N10.a = 0));`	false
	`LTLSPEC G (((N1.a = 0 & (N2.a = 1 & N3.a = 0)) -> F (N10.a = 0 & pInS)) | !pInS)`	
	`SPEC AG (((N1.a = 0 & (N2.a = 1 & N3.a = 0)) -> EF (N10.a = 0 & pInS)) | !pInS)`	
4	`always ((N1.a = 0  and  (N2.a = 1  and  N3.a = 1))  implies (eventually  N10.a = 1));`	true
	`LTLSPEC G (((N1.a = 0 & (N2.a = 1 & N3.a = 1)) -> F (N10.a = 1 & pInS)) | !pInS)`	
	`SPEC AG (((N1.a = 0 & (N2.a = 1 & N3.a = 1)) -> EF (N10.a = 1 & pInS)) | !pInS)`	
5	`always ((N1.a = 1  and  (N2.a = 0  and  N3.a = 0))  implies (eventually  N10.a = 0));`	false
	`LTLSPEC G (((N1.a = 1 & (N2.a = 0 & N3.a = 0)) -> F (N10.a = 0 & pInS)) | !pInS)`	
	`SPEC AG (((N1.a = 1 & (N2.a = 0 & N3.a = 0)) -> EF (N10.a = 0 & pInS)) | !pInS)`	
6	`always ((N1.a = 1  and  (N2.a = 0  and  N3.a = 1))  implies (eventually  N10.a = 1));`	true
	`LTLSPEC G (((N1.a = 1 & (N2.a = 0 & N3.a = 1)) -> F (N10.a = 1 & pInS)) | !pInS)`	
	`SPEC AG (((N1.a = 1 & (N2.a = 0 & N3.a = 1)) -> EF (N10.a = 1 & pInS)) | !pInS)`	
7	`always ((N1.a = 1  and  (N2.a = 1  and  N3.a = 0))  implies (eventually  N10.a = 0));`	false
	`LTLSPEC G (((N1.a = 1 & (N2.a = 1 & N3.a = 0)) -> F (N10.a = 0 & pInS)) | !pInS)`	
	`SPEC AG (((N1.a = 1 & (N2.a = 1 & N3.a = 0)) -> EF (N10.a = 0 & pInS)) | !pInS)`	
8	`always ((N1.a = 1  and  (N2.a = 1  and  N3.a = 1))  implies (eventually  N10.a = 1));`	true
	`LTLSPEC G (((N1.a = 1 & (N2.a = 1 & N3.a = 1)) -> F (N10.a = 1 & pInS)) | !pInS)`	
	`SPEC AG (((N1.a = 1 & (N2.a = 1 & N3.a = 1)) -> EF (N10.a = 1 & pInS)) | !pInS)`	
9	`always ((N1.a = 1) or ((N2.a = 1) or (N3.a = 1))) implies (eventually ((N5.a > 0) or (N9.a > 0)));`	true
	`LTLSPEC G ((N1.a = 1 | (N2.a = 1 | N3.a = 1)) -> F (N5.a > 0 | N9.a > 0))`	
	`SPEC AG ((N1.a = 1 | (N2.a = 1 | N3.a = 1)) -> EF (N5.a > 0 | N9.a > 0))`	
10	`always ((N1.a = 1) or (N3.a = 1)) implies (eventually (N8.a = 1));`	true
	`LTLSPEC G ((N1.a = 1 | N3.a = 1) -> F N8.a = 1)`	
	`SPEC AG ((N1.a = 1 | N3.a = 1) -> EF N8.a = 1)`	

Table 7: Properties of the logical gate model, $k\Pi_{1,p}$

5 Conclusions

In this paper is investigated the process of mapping some of the most representative classes of SN P systems into kP systems, pointing to how different syntactic features, such as alphabet, format of different rules or synapses linking neurons, as well as semantic conditions attached to rule application – rules with or with no delay, constraints on selecting them (based on regular expressions, threshold, electrical charges, strategies for execution), or to relationships between various elements of the system (spikes and anti-spikes, potential and threshold, multisets of electrical charges) have an impact on the complexity aspects of this process. In this respect it has been pointed out that the graphs associated with the structure of the two P systems and the number of compartments vs the number of neurons are almost the same in all the cases. Some complexity metrics measuring the size of the alphabet, $Symb$, overall number of rules, $Rule$, and execution time, $Time$, provide distinct measurements for syntactic and semantic features mentioned above. Another important aspect that is associated with the execution time is provided by the execution strategies associated with compartments of various kP systems obtained through mapping. As the $Time$ measure is quite the same in most of the cases for the two P systems engaged in mapping, the execution strategy reveals new insights related to this complexity metric. All these complexity results and the connections with their mapping theoretical underpinning are summarized in Sections 3.1.1 and 3.2.3.

One direct impact of these investigations connecting SN P systems and kP systems is on extending verification and testing procedures, already in place for kP systems, to various classes of SN P systems. In the context of an application based on an extended SN P system with no delay, Π_e, testing sequences generated from the kP system, $k\Pi_e$ are applied straight away to test the application (based on Π_e) and the model checking verification procedure developed for $k\Pi_e$ is used to verify the correctness of Π_e model.

Some future research avenues coming out of this work are (i) mapping of other SN P systems, with a different spectrum of features, such as those with dynamic structure or with plasticity; (ii) mapping some of the most complex applications based on SN P systems into kP systems and using testing and verification methods to validate them; (iii) developing further testing methods that are able to reveal other types of errors than those presented here; (iv) providing an integrated tool supporting the development, testing and verification of case studies and applications for these models.

Acknowledgements

The authors of this paper would like to thank all the anonymous reviewers for their effort and time spent in carefully reading the manuscript and providing constructive and useful comments and suggestions that significantly contributed to the paper improvement. Florentin Ipate's and Ionuţ Mihai Niculescu's research was supported by the European Regional Development Fund, Competitiveness Operational Program 2014-2020 through project IDBC (code SMIS 2014+: 121512). The other contributors have been partially supported by the Royal Society grant IESR3213176, 2022-2025.

References

[1] Bakir, M.E., Ipate, F., Konur, S., Mierlă, L., Niculescu, I.-M.: Extended simulation and verification platform for kernel P systems. In: Gheorghe M. et al (ed.) 15th Int. Conference on Membrane Computing, LNCS 8961, pp. 158–178 (2014)

[2] Cabarle, F.G.C., Adorna, H.N., Jiang, M., Zeng, X.: Spiking neural P systems with scheduled synapses. IEEE Transactions on Nanobioscience **16**(8), 792–801 (2017)

[3] Cabarle, F.G.C., Adorna, H.N., Pérez-Jiménez, M.J., Song, T.: Spiking neural P systems with structural plasticity. Neural Computing and Applications **26**(8), 1905–1917 (2015)

[4] Chen, H., Ishdorj, T.O., Ionescu, M., Păun, A., Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems with extended rules: universality and languages. Natural Computing **7**(2), 147–166 (2008)

[5] Chen, Y., Chen, Y., Zhang, G., Paul, P., Wu, T., Zhang, X., Rong, H., Ma, X.: A survey of learning spiking neural P systems and a novel instance. International Journal of Unconventional Computing **16**(2–3), 173–200 (2021)

[6] Frisco, P., Pérez-Jiménez, M.J., Gheorghe, M. (eds.): Applcations of Membrane Computing in Systems and Synthetic Biology. Springer, Verlag (2014)

[7] Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S.: Kernel P systems modelling, testing and verification - sorting case study. In: Leporati, A. et al (ed.) 17th Int. Conference on Membrane Computing, LNCS 10105, pp. 233–250. Cham (2017)

[8] Gheorghe, M., Ceterchi, R., Ipate, F., Konur, S., Lefticaru, R.: Kernel P systems: from modelling to verification and testing. Theoretical Computer Science **724**, 45–60 (2018). URL http://hdl.handle.net/10454/11720

[9] Gheorghe, M., Ipate, F., Dragomir, C., Mierlă, L., Valencia-Cabrera, L., García-Quismondo, M., Pérez-Jiménez, M.J.: Kernel P systems - Version I. 11th Brainstorming Week on Membrane Computing pp. 97–124 (2013). URL http://www.gcn.us.es/files/11bwmc/097_gheorghe_ipate.pdf

[10] Gheorghe, M., Ipate, F., Konur, S.: Testing based on identifiable P systems using cover automata and X-machines. Information Sciences **372**, 565—-578 (2016)

[11] Gheorghe, M., Konur, S., Ipate, F., Mierlă, L., Bakir, M.E., Stannett, M.: An integrated model checking toolset for kernel P systems. In: Rozenberg, G. et al (ed.) 16th Int. Conference on Membrane Computing, LNCS 9504, pp. 153–170. Springer (2015)

[12] Gheorghe, M., Lefticaru, R., Konur, S., Niculescu, I.M., Adorna, H.N.: Spiking neural P systems: matrix representation and formal verification. Journal of Membrane Computing **3**(2), 133–148 (2021). DOI 10.1007/s41965-021-00075-1

[13] Ibarra, O.H., Păun, A., Păun, Gh., Rodríguez-Patón, A., Sosik, P., Woodworth, S.: Normal forms for spiking neural P systems. Theoretical Computer Science **372**, 196–217 (2007)

[14] Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae **71**(2–3), 279–308 (2006)

[15] Ipate, F., Gheorghe, M.: Finite state based testing of P systems. Natural Computing **8**(4), 833–846 (2009)

[16] Ipate, F., Gheorghe, M.: A model learning based testing approach for spiking

neural P systems. Theoretical Computer Science **924**, 1–16 (2022)

[17] Ipate F. Niculescu, I., Lefticaru, R., Konur, S., Gheorghe, M.: A model learning based testing approach for kernel P systems. Theoretical Computer Science **965**, 113975 (2023)

[18] Konur, S., Gheorghe, M., Dragomir, C., Ipate, F., Krasnogor, N.: Conventional verification for unconventional computing: a genetic XOR gate example. Fundamenta Informaticae **134**(1–2), 97–110 (2014)

[19] Konur, S., Gheorghe, M., Dragomir, C., Mierlă, L., Ipate, F., Krasnogor, N.: Qualitative and quantitative analysis of systems and synthetic biology constructs using P systems. ACS Synthetic Biology **4**(1), 83–92 (2015)

[20] Konur, S., Mierlă, L., Ipate, F., Gheorghe, M.: kPWorkbench: A software suit for membrane systems. SoftwareX **11**, 100407 (2020)

[21] Lefticaru, R., Gheorghe, M., Ipate, F.: An empirical evaluation of P system testing techniques. Natural Computing **10**(1), 151—-165 (2016)

[22] Leporati, A., Mauri, G., Zandron, C.: Spiking neural P systems: main ideas and results. Natural Computing **21**(4), 629–649 (2022). DOI https://doi.org/10.1007/s11047-022-09917-y

[23] Li, Y., Song, B., Zeng, X.: Spiking neural P systems with weights and delays on synapses. Theoretical Computer Science **968**, 114028 (2023)

[24] Liu, Y., Zhao, Y.: Spiking neural P systems with lateral inhibition. Neural Networks **167**, 36–49 (2023)

[25] Lv, Z., Bao, T., Zhou, N., Peng, H., Huang, A., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Spiking neural P systems with extended channel rules. International Journal of Neural Systems **31**(1), 2050049 (2021)

[26] Macababayao, I.C.H., Cabarle, F.G.C., de la Cruz, R.T., Zeng, X.: Normal forms for spiking neural P systems and some of its variants. Information Sciences **595**, 344–363 (2022)

[27] Niculescu, I.M.: Logical gate (2025). URL https://github.com/Kernel-P-Systems/kPWorkbench/tree/Mapping-SNPS-Into-kPS

[28] Pan, L., Păun, Gh.: Spiking neural P systems with anti-spikes. International Journal of Computers, Communications & Control **IV**(3), 273–282 (2009)

[29] Pan, L., Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems with neuron division and budding. Science China Information Sciences **54**, 1596–1607 (2011)

[30] Pan, L., Wang, J., Hoogeboom, H.: Spiking neural P systems with astrocytes. Neural Computation **24**, 805–825 (2012)

[31] Pan, L., Wu, T., Zhang, Z.: A bibliography of spiking neural P systems. Bulletin of the International Membrane Computing Society (I M C S) **1**(1), 63–78 (2016)

[32] Pan, L., Zeng, X., Zhang, X., Jiang, Y.: Spiking neural P systems with weighted synapses. Neural Processing Letters **35**(1), 13–27 (2012)

[33] Pang, S., Wang, M., Qiao, S., Wang, X., Chen, H.: Fault diagnosis for service composition by spiking neural P systems with colored spikes. Chinese Journal of Electronis **28**(5), 1033–1040 (2019)

[34] Păun, Gh.: Computing with membranes. Tech. rep., Turku Centre for Computer Science (1998). URL http://tucs.fi/publications/view/?pub_id=tPaun98a

[35] Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000). DOI 10.1006/jcss.1999.1693. URL https://doi.org/10.1006/jcss.1999.1693
[36] Păun, Gh.: Membrane Computing - An Introduction. Springer, Verlag (2002)
[37] Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
[38] Păun, Gh., Wu, T., Zhang, Z.: Open problems, research topics, recent results on numerical and spiking neural P systems (The 'Curtea de Argeş 2015 series'). In: Proceedings of Fourteenth Brainstorming Week on Membrane Computing, pp. 285–300. Sevilla, Spain: Fenix Editora (2016)
[39] Peng, H., Bao, T., Luo, X., Wang, J., Song, X., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Dendrite P systems. Neural Networks **127**, 110–120 (2020)
[40] Peng, H., Li, B., Wang, J., Song, X., Wang, T., Valencia-Cabrera, L., Pérez-Hurtado, I., Riscos-Núñez, A., Pérez-Jiménez, M.J.: Spiking neural P systems with inhibitory rules. Knowledge-Based Systems **188**, 105064 (2020)
[41] Peng, H., Lv, Z., Li, B., Luo, X., Wang, J., Song, X., Wang, T., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Nonlinear spiking neural P systems. International Journal of Neural Systems **30**(10), 2050008 (2020)
[42] Peng, H., Wang, J.: Coupled neural P systems. IEEE Transactions on Neural Networks and Learning Systems **30**(6), 1672–1682 (2019)
[43] Peng, H., Wang, J.: Advanved Spiking Neural P Systems - Models and Applications. Springer, Verlag (2024)
[44] Peng, H., Wang, J., Pérez-Jiménez, M.J., Riscos-Núñez, A.: Dynamic threshold neural P systems. Knowledge-Based Systems **163**, 875–884 (2019)
[45] Peng, H., Wang, J., Pérez-Jiménez, M.J., Wang, H., Shao, J., Wang, T.: Fuzzy reasoning spiking neural P system for fault diagnosis. Information Sciences **235**, 106–118 (2013)
[46] Peng, H., Yang, J., Wang, J., Wang, T., Sun, Z., Song, X., Luo, X., Huang, X.: Spiking neural P systems with multiple channels. Neural Networks **95**, 66–71 (2017)
[47] Rong, H., Wu, T., Pan, L., Zhang, G.: Spiking neural P systems: Theoretical results and applications. In: Graciani, C. et al. (ed.) Enjoying Natural Computing, LNCS 11270, pp. 256–268 (2018)
[48] Song, B., Li, K., Orellana-Martín, D., Pérez-Jiménez, M.J., Hurtado, I.P.: A survey of nature-inspired computing: Membrane computing. ACM Computing Surveys **54**(1), 629–649 (2021). DOI https://doi.org/10.1145/3431234
[49] Song, T., Gong, F., Liu, X., Zhao, Y., Zhang, X.: Spiking neural P systems with white hole neurons. IEEE Transactions on Nanobioscience **15**(7), 666–673 (2016)
[50] Song, T., Pan, L.: Spiking neural P systems with request rules. Neurocomputing **193**, 2816–2829 (2016)
[51] Song, T., Pan, L., Păun, Gh.: Spiking neural P systems with rules on synapses. Theoretical Computer Science **529**, 82–95 (2014)
[52] Song, T., Rodríguez-Patón, A., Zheng, P., Zeng, X.: Spiking neural P systems with colored spikes. IEEE Transactions on Cognitive and Development Systems **10**(4), 2816–2829 (2017)

[53] Ţurlea, A., Gheorghe, M., Ipate, F., Konur, S.: Search-based testing in membrane computing. Journal of Membrane Computing **1**(4), 565—-578 (2019)

[54] Verlan, S., Freund, R., Alhazov, A., Ivanov, S., Pan, L.: A formal framework for spiking neural P systems. Journal of Membrane Computing **2**(4), 355–368. (2020). DOI 10.1007/s41965-021-00075-1

[55] Wang, J., Hoogeboom, H., Pan, L., Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems with weights. Neural Compution **22**(10), 2615–2646 (2010)

[56] Wu, T., Neri, F., Pan, L.: On the tuning of the computation capability of spiking neural membrane systems with communication on request. International Journal of Neural Systems **32**(8), 2250037 (2022)

[57] Wu, T., Pan, L., Tan, K.C.: Numerical spiking neural P systems. IEEE Transactions on Neural Networks and Learning Systems **32**(6), 2816–2829 (2021)

[58] Wu, T., Păun, A., Zhang, Z., Pan, L.: Spiking neural P systems with polarizations. IEEE Transactions on Neural Networks and Learning Systems **29**(8), 3349–3360 (2018)

[59] Zhang, G., Pérez-Jiménez, M.J., Gheorghe, M.: Real-life Applications with Membrane Computing. Springer, Verlag (2017)

[60] Zhang, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Verlan, S., Konur, S., Hinze, T., Gheorghe, M.: Membrane Computing Models: Implementations. Springer, Verlag (2021)

[61] Zhang, G., Verlan, S., Wu, T., Cabarle, F., Xue, J., Orellan-Martín, D., Dong, J., Valencia-Cabrera, L., Pérez-Jiménez, M.J.: Spiking Neural P Systems - Theory, Applications and Implementations. Springer, Verlag (2024)

[62] Zhang, X., Pan, L., Păun, A.: On the universality of axon P systems. IEEE Transactions on Neural Networks and Learning Systems **26**(11), 2816–2829 (2015)